Open Group Technical Standard

X Window System (X11R6):
Protocol

The Open Group

Open Group Technical Standard

X Window System (X11R6): Protocol

Document Number: C903

Published in the U.K. by The Open Group, June 1999.

Any comments relating to the material contained in this document may be submitted to:

The Open Group
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

OGSpecs@opengroup.org

Contents

Preface

The Open Group

The Open Group is the leading vendor-neutral, international consortium for buyers and suppliers of technology. Its mission is to cause the development of a viable global information infrastructure that is ubiquitous, trusted, reliable, and as easy-to-use as the telephone. The essential functionality embedded in this infrastructure is what we term the *IT DialTone*. The Open Group creates an environment where all elements involved in technology development can cooperate to deliver less costly and more flexible IT solutions.

Formed in 1996 by the merger of the X/Open Company Ltd. (founded in 1984) and the Open Software Foundation (founded in 1988), The Open Group is supported by most of the world's largest user organizations, information systems vendors, and software suppliers. By combining the strengths of open systems specifications and a proven branding scheme with collaborative technology development and advanced research, The Open Group is well positioned to meet its new mission, as well as to assist user organizations, vendors, and suppliers in the development and implementation of products supporting the adoption and proliferation of systems which conform to standard specifications.

With more than 200 member companies, The Open Group helps the IT industry to advance technologically while managing the change caused by innovation. It does this by:

- Consolidating, prioritizing, and communicating customer requirements to vendors
- Conducting research and development with industry, academia, and government agencies to deliver innovation and economy through projects associated with its Research Institute
- Managing cost-effective development efforts that accelerate consistent multi-vendor deployment of technology in response to customer requirements
- Adopting, integrating, and publishing industry standard specifications that provide an essential set of blueprints for building open information systems and integrating new technology as it becomes available
- Licensing and promoting the Open Brand, represented by the ''X'' Device, that designates vendor products which conform to Open Group Product Standards
- Promoting the benefits of the IT DialTone to customers, vendors, and the public

The Open Group operates in all phases of the open systems technology lifecycle including innovation, market adoption, product development, and proliferation. Presently, it focuses on seven strategic areas: open systems application platform development, architecture, distributed systems management, interoperability, distributed computing environment, security, and the information superhighway. The Open Group is also responsible for the management of the UNIX trademark on behalf of the industry.

Development of Product Standards

This process includes the identification of requirements for open systems and, now, the IT DialTone, development of Technical Standards (formerly CAE and Preliminary Specifications) through an industry consensus review and adoption procedure (in parallel with formal standards work), and the development of tests and conformance criteria.

This leads to the preparation of a Product Standard which is the name used for the documentation that records the conformance requirements (and other information) to which a vendor may register a product.

The ''X'' Device is used by vendors to demonstrate that their products conform to the relevant Product Standard. By use of the Open Brand they guarantee, through the Open Brand Trade Mark License Agreement (TMLA), to maintain their products in conformance with the Product Standard so that the product works, will continue to work, and that any problems will be fixed by the vendor.

Open Group Publications

The Open Group publishes a wide range of technical documentation, the main part of which is focused on development of Technical Standards and product documentation, but which also includes Guides, Snapshots, Technical Studies, Branding and Testing documentation, industry surveys, and business titles.

There are several types of specification:

- *Technical Standards* (formerly *CAE Specifications*)

  The Open Group Technical Standards form the basis for our Product Standards. These Standards are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement a Technical Standard can enjoy the benefits of a single, widely supported industry standard. Where appropriate, they can demonstrate product compliance through the Open Brand. Technical Standards are published as soon as they are developed, so enabling vendors to proceed with development of conformant products without delay.

- *CAE Specifications*

  CAE Specifications and Developers' Specifications published prior to January 1998 have the same status as Technical Standards (see above).

- *Preliminary Specifications*

  Preliminary Specifications have usually addressed an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations. They are published for the purpose of validation through implementation of products. A Preliminary Specification is as stable as can be achieved, through applying The Open Group's rigorous development and review procedures.

  Preliminary Specifications are analogous to the *trial-use* standards issued by formal standards organizations, and developers are encouraged to develop products on the basis of them. However, experience through implementation work may result in significant (possibly upwardly incompatible) changes before its progression to becoming a Technical Standard. While the intent is to progress Preliminary Specifications to corresponding Technical Standards, the ability to do so depends on consensus among Open Group members.

- *Consortium and Technology Specifications*

  The Open Group publishes specifications on behalf of industry consortia. For example, it publishes the NMF SPIRIT procurement specifications on behalf of the Network Management Forum. It also publishes Technology Specifications relating to OSF/1, DCE, OSF/Motif, and CDE.

  Technology Specifications (formerly AES Specifications) are often candidates for consensus review, and may be adopted as Technical Standards, in which case the relevant Technology Specification is superseded by a Technical Standard.

In addition, The Open Group publishes:

- *Product Documentation*

  This includes product documentation—programmer's guides, user manuals, and so on—relating to the Pre-structured Technology Projects (PSTs), such as DCE and CDE. It also includes the Single UNIX Documentation, designed for use as common product documentation for the whole industry.

- *Guides*

  These provide information that is useful in the evaluation, procurement, development, or management of open systems, particularly those that relate to the Technical Standards or Preliminary Specifications. The Open Group Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming conformance to a Product Standard.

- *Technical Studies*

  Technical Studies present results of analyses performed on subjects of interest in areas relevant to The Open Group's Technical Program. They are intended to communicate the findings to the outside world so as to stimulate discussion and activity in other bodies and the industry in general.

Versions and Issues of Specifications

As with all *live* documents, Technical Standards and Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.
- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Corrigenda

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published on the World-Wide Web at *http://www.opengroup.org/corrigenda*.

Ordering Information

Full catalogue and ordering information on all Open Group publications is available on the World-Wide Web at *http://www.opengroup.org/pubs*.

This Technical Standard

This Technical Standard is one of a set of Technical Standards (see above) defining the X Window System.

This Technical Standard specifies the X Protocol for X11R6.4 and subsequent releases of the X Window System.

The X Window System Registry

The X Window System Registry is a registry of certain X-related items, to help avoid conflicts and share such items. The Open Group maintains the registry as a service to users of the X Window System. Requests to register items, or questions about registration, should be addressed to:

xregistry@x.org

Electronic mail is acknowledged upon receipt. Please allow up to 4 weeks for a formal response to registration and enquiries.

All registered items must have the postal address of someone responsible for the item or a reference to a document describing the item and the postal address of where to write to obtain the document.

The X Window System Technical Standards indicate that the following items are subject to registration:

- Names of non-standard character encodings
- Prefixes for private font names (which follow the FONT keyword in the **BDF** specification (included in the **X Window System (X11R6): Fonts and Text** Technical Standard))
- Transport-specific names other than those specified by The Open Group
- The *FontNameRegistry* component of a *FontName* in the **XLFD** specification (included in the **X Window System (X11R6): Fonts and Text** Technical Standard)
- Values of CHARSET_ENCODING, CHARSET_REGISTRY, FONT_TYPE, and FOUNDRY in the **XLFD** specification (included in the **X Window System (X11R6): Fonts and Text** Technical Standard)
- New values of AXIS_TYPES in the **XLFD** specification (included in the **X Window System (X11R6): Fonts and Text** Technical Standard), and the semantics of each
- Window manager protocols specified by the WM_PROTOCOLS property
- Device-dependent color spaces represented by the **XcmsColorSpace** structure
- New values of the **XIC** and **XOC** structures for the X Input Context and X Output Context
- In the Input Device Extension (XINPUT), new input classes
- In the Display Manager Control Protocol (XDMCP) Extension, values for the *ManufacturerID* parameter of the *Manage* message
- In the X Input Method (XIM) Extension, new values of XIM_ENCODING_NEGOTIATION; also, opcodes and packet names for additions to this extension
- The *organization* component of names of algorithms in the Low Bandwidth Extension (LBX)
- The names of system counters, as described in the Synchronization (XSYNC) Extension

Trademarks

Motif®, OSF/1®, UNIX®, and the ‘‘X Device’’® are registered trademarks and IT DialTone™ and The Open Group™ are trademarks of The Open Group in the U.S. and other countries.

X Window System™ is a trademark of The Open Group.

Acknowledgements

The Open Group gratefully acknowledges the following Working Group members for their valuable contribution to the development of this Technical Standard:

Art Barstow
Brian Bobryk
Linda Cain
Peter Daifuku
Vic Goddard
Cathy Hughes
Kaleb S. Keithley
Finnbarr P. Murphy
Spike
Steve Swales
Jeff Walls

Referenced Documents

The following documents are referenced in this Technical Standard:

- ISO 2022: 1986, Information Processing — ISO 7-bit and 8-bit Coded Character Sets — Coded Extension Techniques.
- ISO 8859-1: 1987, Information Processing — 8-bit Single-byte Coded Graphic Character Sets — Part 1: Latin Alphabet No. 1.
- ISO / IEC 9899: 1990, Programming Languages — C, including:
  Amendment 1: 1995 (E), C Integrity (Multibyte Support Extensions (MSE) for ISO C).
- ISO / IEC 9945-1: 1996, Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language] (identical to ANSI / IEEE Std 1003.1-1996). Incorporating ANSI / IEEE Stds 1003.1-1990, 1003.1b-1993, 1003.1c-1995 and 1003.1i-1995.
- Technical Standard, estimated late 1999, X Window System (X11R6): C-Language Library (Xlib) (C904), published by The Open Group.
- Technical Standard, estimated early 2000, X Window System (X11R6): Extensions (C906), published by The Open Group.
- Technical Standard, estimated early 2000, X Window System (X11R6): Fonts and Text (C907), published by The Open Group.
- Technical Standard, estimated late 1999, X Window System (X11R6): Toolkit (C905), published by The Open Group.

Chapter 1

Introduction

1.1 This Issue (X11R6)

Technical Differences

Major differences from the previous issue of this Technical Standard are as follows:

- Keysyms are provided in Appendix A for the Thai (13), Korean (14), ISO Latin-5 (15), ISO Latin-6 (16), ISO Latin-7 (17), ISO Latin-8 (18), ISO Latin-9 (19), and Currency (254) character sets.
- In Section 2.5 on page 15 and Section 5.4 on page 101, the server can now return a response to the client indicating that further authentication is required. The *success* field is a tri-state field with the values *Failed*, *Success*, or *Authenticate*.
- A code point for the euro currency has been added.

Editorial Changes

Other changes are organizational and editorial and are not intended to suggest substantive changes. For example, events are now presented in alphabetical order, as requests always have been. The *ButtonPress*, *ButtonRelease*, *KeyPress*, *KeyRelease*, and *MotionNotify* events are presented in three entries instead of in a single entry.

1.2 Terminology

The following terms are used in this document:

can
: Describes a permissible optional feature or behavior available to the user or application. The feature or behavior is mandatory for an implementation that conforms to this document. An application can rely on the existence of the feature or behavior.

implementation-dependent
: (Same meaning as *implementation-defined*.) Describes a value or behavior that is not defined by this document but is selected by an implementor. The value or behavior may vary among implementations that conform to this document. An application should not rely on the existence of the value or behavior. An application that relies on such a value or behavior cannot be assured to be portable across conforming implementations.

  The implementor shall document such a value or behavior so that it is used correctly by an application.

legacy
: Describes a feature or behavior that is being retained for compatibility with older applications, but which has limitations which make it inappropriate for developing portable applications. New applications should use alternative means of obtaining equivalent functionality.

may
Describes a feature or behavior that is optional for an implementation that conforms to this document. An application should not rely on the existence of the feature or behavior. An application that relies on such a feature or behavior cannot be assured to be portable across conforming implementations.

To avoid ambiguity, the opposite of *may* is expressed as *need not*, instead of *may not*.

must
Describes a feature or behavior that is mandatory for an application or user. An implementation that conforms to this document shall support this feature or behavior.

shall
Describes a feature or behavior that is mandatory for an implementation that conforms to this document. An application can rely on the existence of the feature or behavior.

should
For an implementation that conforms to this document, describes a feature or behavior that is recommended but not mandatory. An application should not rely on the existence of the feature or behavior. An application that relies on such a feature or behavior cannot be assured to be portable across conforming implementations.

For an application, describes a feature or behavior that is recommended programming practice for optimum portability.

undefined
Describes the nature of a value or behavior not defined by this document which results from use of an invalid program construct or invalid data input.

The value or behavior may vary among implementations that conform to this document. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.

unspecified
Describes the nature of a value or behavior not specified by this document which results from use of a valid program construct or valid data input.

The value or behavior may vary among implementations that conform to this document. An application should not rely on the existence or validity of the value or behavior. An application that relies on any particular value or behavior cannot be assured to be portable across conforming implementations.

will
Same meaning as *shall*; *shall* is the preferred term.

1.3 Syntactic Conventions

This Technical Standard uses the following syntactic conventions.

- The syntax {...} encloses a set of alternatives.
- The syntax [...] encloses a set of structure components.
- Types are in uppercase, such as BITMASK.
- Parameter names are in italic lowercase, such as *length*. Alternative values appear in italics and mixed case, such as *AsyncPointer*. (Italics is not used in the protocol encoding.)
- Errors are in uppercase, enclosed in square brackets; for example, [ERROR].

1.3.1 Protocol Descriptions

Requests are described in the following format:

RequestName

arg1: type1
...
argN: typeN
→
result1: type1
...
resultM: typeM

Errors: *kind1*, ..., *kindK*

If no → is present in the description, then the request has no reply (it is asynchronous), although errors may still be reported. If →+ is used, then one or more replies can be generated for a single request.

1.3.2 Events

Events are described in the following format:

EventName

value1: type1
...
valueN: typeN

1.3.3 Protocol Encoding

The protocol encodings use the following syntactic conventions:

- All numbers are in decimal, unless prefixed with #0x, in which case they are in hexadecimal (base 16).
- The syntax used to describe requests, replies, errors, events, and compound types is:

  NameofThing
  encode-form
  ...
  encode-form

Each encode-form describes a single component.

- For components described in the protocol as:

  name: TYPE

  the encode-form is:

N	TYPE	name

  N is the number of bytes occupied in the data stream, and TYPE is the interpretation of those bytes. For example,

  depth: CARD8

  becomes:

1	CARD8	depth

- For components with a static numeric value the encode-form is:

N	value	name

  The value is always interpreted as an N-byte unsigned integer. For example, the first two bytes of a *Window* error are always 0 (indicating an error in general) and 3 (indicating the *Window* error in particular):

1	0	Error
1	3	code

- For components described in the protocol as:

  name: {*Name1*, ..., *NameN*}

  the encode-form is:

N		name
	value1	Name1
	...	
	valueN	NameN

  The value is always interpreted as an N-byte unsigned integer. The size of N is sometimes larger than that strictly required to encode the values. For example:

  class: {*InputOutput*, *InputOnly*, *CopyFromParent*}

  becomes:

2		class
	0	CopyFromParent
	1	InputOutput
	2	InputOnly

- For components described in the protocol as:

  NAME: TYPE or *Alternative1* ...or *AlternativeN*

  the encode-form is:

N	TYPE	NAME
	value1	Alternative1
	...	
	valueN	AlternativeN

The alternative values are guaranteed not to conflict with the encoding of TYPE. For example:

destination: WINDOW or *PointerWindow* or *InputFocus*

becomes:

4	WINDOW	destination
	0	PointerWindow
	1	InputFocus

- For components described in the protocol as:

  value-mask: BITMASK

  the encode-form is:

N	BITMASK	value-mask
	mask1	mask-name1
	...	
	maskI	mask-nameN

  The individual bits in the mask are specified and named, and N is 2 or 4. The most-significant bit in a BITMASK is reserved for use in defining chained (multi-word) bitmasks, as extensions augment existing core requests. The precise interpretation of this bit is unspecified, although a probable mechanism is that a 1-bit indicates that another N bytes of bitmask follows, with bits within the overall mask still interpreted from least-significant to most-significant with an N-byte unit, with N-byte units interpreted in stream order, and with the overall mask being byte-swapped in individual N-byte units.

- For LISTofVALUE encodings, the request is followed by a section of the form:

  VALUE
  encode-form
  ...
  encode-form

  listing an encode-form for each VALUE. The NAME in each encode-form keys to the corresponding BITMASK bit. The encoding of a VALUE always occupies 4 bytes, but the number of bytes specified in the encoding-form indicates how many of the least-significant bytes are actually used; the remaining bytes are unused and their values do not matter.

  In various cases, the number of bytes occupied by a component is specified by a lowercase single-letter variable name instead of a specific numeric value, and often some other component has its value specified as a simple numeric expression involving these variables. Components specified with such expressions are always interpreted as unsigned integers. The scope of such variables is always just the enclosing request, reply, error, event, or compound type structure. For example:

2	3+n	request length
4n	LISTofPOINT	points

- For unused bytes (the values of the bytes are unspecified and do not matter), the encode-form is:

N	unused

  If the number of unused bytes is variable, the encode-form is:

p	unused, *p*=pad(*E*)

where *E* is some expression, and pad(*E*) is the number of bytes needed to round E up to a multiple of 4:

pad(*E*) = (4–(*E* mod 4)) mod 4

Chapter 2

Protocol Overview

Clients and servers use some form of reliable communication to exchange information. When the client and the server reside on different machines, a reliable duplex byte stream is required. The X protocol specifies a byte-ordering (see Section 2.5).

2.1 Data Formats

Clients send requests to servers. Servers respond by sending replies or errors to the client. A server can also send an event to the client asynchronously to any request. A client request can specify whether the client wishes to receive certain types of event.

Unused bytes in any data item are not guaranteed to be 0.

2.1.1 Request Format

Every request contains an 8-bit major opcode and a 16-bit length field expressed in units of 4 bytes. Every request consists of 4 bytes of a header (containing the major opcode, the length field, and a data byte) followed by 0 or more additional bytes of data. The length field defines the total length of the request, including the header. The length field in a request must equal the minimum length required to contain the request. If the specified length is smaller or larger than the required length, an error is generated.

Major opcodes 128 through 255 are reserved for extensions. Extensions are intended to contain multiple requests, so extension requests typically have an additional minor opcode encoded in the ‘‘spare’’ data byte in the request header. However, the placement and interpretation of this minor opcode and of all other fields in extension requests are not defined by the core X protocol.

Every request on a given connection is implicitly assigned a sequence number, starting with one, that is used in replies, errors, and events.

2.1.2 Reply Format

Every reply contains a 32-bit length field expressed in units of 4 bytes. Every reply consists of 32 bytes followed by 0 or more additional bytes of data, as specified in the length field. Every reply also contains the least-significant 16 bits of the sequence number of the corresponding request.

2.1.3 Error Format

Error reports are 32 bytes long. Every error includes an 8-bit error code. Error codes 128 through 255 are reserved for extensions. Every error also includes the major and minor opcodes of the failed request and the least-significant 16 bits of the sequence number of the request. For more details, see Section 2.7.

2.1.4 Event Format

Events are 32 bytes long. Every event contains an 8-bit type code. The most-significant bit in this code is set if the event was generated from a *SendEvent* request (see **SendEvent** on page 73). Event codes 64 through 127 are reserved for extensions, although the core protocol does not define a mechanism for selecting interest in such events.

Every core event except *KeymapNotify* (see **KeymapNotify** on page 88) also contains the least-significant 16 bits of the sequence number of the last request issued by the client that was (or is currently being) processed by the server.

2.2 Types

The X protocol uses the data types described below. The protocol encoding for each type, and additional information related to transmission of the type, is defined in Section 5.1 on page 95.

Name	Value
LISTof*TYPE*	A type name of the form LISTof*TYPE* means a counted list of elements of the type *TYPE*. A separate field may be provided to indicate the number of elements (and the data type of this field is specified separately), or there may be a predefined number of elements. Except where explicitly noted, zero-length lists are legal.
SETof*TYPE*	A set indicates the presence or absence of each valid value of the data type *TYPE*.
BITMASK LISTofVALUE	Various requests contain parameters of the form: *value-mask*: BITMASK *value-list*: LISTofVALUE These let the client specify a subset of a heterogeneous collection of optional parameters. The *value-mask* specifies which parameters are to be provided; each such parameter is assigned a unique bit position. The representation of the BITMASK typically contains more bits than there are defined parameters. The unused bits in *value-mask* must be 0 (or the server generates a [VALUE] error). The *value-list* contains one value for each bit set to 1 in the mask, from least-significant to most-significant bit in the mask. Each value is represented with 4 bytes, but the actual value occupies only the least-significant bytes as required. The values of the unused bytes do not matter.
OR	A type of the form ‘‘T1 or ... or T*n*’’ means the union of the indicated types. A single-element type is given as the element without enclosing braces.
WINDOW	32-bit value (top three bits guaranteed to be 0)
PIXMAP	32-bit value (top three bits guaranteed to be 0)
CURSOR	32-bit value (top three bits guaranteed to be 0)
FONT	32-bit value (top three bits guaranteed to be 0)
GCONTEXT	32-bit value (top three bits guaranteed to be 0)
COLORMAP	32-bit value (top three bits guaranteed to be 0)
DRAWABLE	WINDOW or PIXMAP
FONTABLE	FONT or GCONTEXT
ATOM	32-bit value (top three bits guaranteed to be 0)
VISUALID	32-bit value (top three bits guaranteed to be 0)
VALUE	32-bit quantity (used only in LISTofVALUE)
BYTE	8-bit value
INT8	8-bit signed integer
INT16	16-bit signed integer
INT32	32-bit signed integer
CARD8	8-bit unsigned integer

Name	Value
CARD16	16-bit unsigned integer
CARD32	32-bit unsigned integer
TEXTITEM8	TEXTELT8 or FONT
TEXTELT8	[*delta*: INT8 *string*: STRING8]
TEXTITEM16	TEXTELT16 or FONT
TEXTELT16	[*delta*: INT8 *string*: STRING16]
TIMESTAMP	CARD32
COLORITEM	[*pixel*: CARD32 *do-red*, *do-green*, *do-blue*: BOOL *red*, *green*, *blue*: CARD16]
BITGRAVITY	{*Forget*, *Static*, *NorthWest*, *North*, *NorthEast*, *West*, *Center*, *East*, *SouthWest*, *South*, *SouthEast*}
WINGRAVITY	{*Unmap*, *Static*, *NorthWest*, *North*, *NorthEast*, *West*, *Center*, *East*, *SouthWest*, *South*, *SouthEast*}
BOOL	{*True*, *False*}
EVENT	{*KeyPress*, *KeyRelease*, *OwnerGrabButton*, *ButtonPress*, *ButtonRelease*, *EnterWindow*, *LeaveWindow*, *PointerMotion*, *PointerMotionHint*, *Button1Motion*, *Button2Motion*, *Button3Motion*, *Button4Motion*, *Button5Motion*, *ButtonMotion*, *Expose*, *VisibilityChange*, *StructureNotify*, *ResizeRedirect*, *SubstructureNotify*, *SubstructureRedirect*, *FocusChange*, *PropertyChange*, *ColormapChange*, *KeymapState*}
POINTEREVENT	{*ButtonPress*, *ButtonRelease*, *EnterWindow*, *LeaveWindow*, *PointerMotion*, *PointerMotionHint*, *Button1Motion*, *Button2Motion*, *Button3Motion*, *Button4Motion*, *Button5Motion*, *ButtonMotion*, *KeymapState*}
DEVICEEVENT	{*KeyPress*, *KeyRelease*, *ButtonPress*, *ButtonRelease*, *PointerMotion*, *Button1Motion*, *Button2Motion*, *Button3Motion*, *Button4Motion*, *Button5Motion*, *ButtonMotion*}
KEYSYM	32-bit value (top three bits guaranteed to be 0)
KEYCODE	CARD8
BUTTON	CARD8
KEYMASK	{*Shift*, *Lock*, *Control*, *Mod1*, *Mod2*, *Mod3*, *Mod4*, *Mod5*}
BUTMASK	{*Button1*, *Button2*, *Button3*, *Button4*, *Button5*}
KEYBUTMASK	KEYMASK or BUTMASK
STRING8	LISTofCARD8
STRING16	LISTofCHAR2B
CHAR2B	[*byte1*, *byte2*: CARD8]
POINT	[*x*, *y*: INT16]
RECTANGLE	[*x*, *y*: INT16, *width*, *height*: CARD16]
ARC	[*x*, *y*: INT16, *width*, *height*: CARD16, *angle1*, *angle2*: INT16]

Name	Value
HOST	[*family*: {*Internet*, *Chaos*} *address*: LISTofBYTE]

The [*x*, *y*] coordinates of a RECTANGLE specify the upper-left corner.

Characters in a STRING16 comprise two bytes that can index into a 2-D matrix;[1] hence, the use of CHAR2B rather than CARD16. The server never byte-swaps CHAR2B quantities.

Fonts should be defined with 2-byte matrix indexing. For fonts constructed with linear indexing, a CHAR2B can be interpreted as a 16-bit number by treating *byte1* as the most-significant byte. Clients should always transmit *byte1* first.

The length, format, and interpretation of a HOST address are specific to the family (see **ChangeHosts** on page 29).

1. **Application Usage:** This corresponds to the JIS/ISO method of indexing 2-byte characters.

2.3 Atoms

Conceptually, atoms are unique names that clients can use to interchange information efficiently. They can be thought of as strings without an encoding. The core X protocol imposes no semantics on these names. Other Technical Standards may specify semantics for them.

A client uses the *InternAtom* protocol request to register a byte sequence with the server. The server returns a 32-bit value (with the top three bits 0) that uniquely represents the byte sequence. The inverse operator is *GetAtomName*.

Predefined Atoms

A predefined atom is an atom that any server recognizes without the client having to intern it. The core X protocol specifies that certain atoms are predefined. For a list of predefined atoms, see Section 5.3 on page 100.

Predefinition is an optimization to eliminate routine *InternAtom* requests from the start-up phase of many client applications. Applications can use any predefined atom as though it had issued an *InternAtom* request successfully.

Predefined atoms are predefined only in the sense of having numeric values, not in the sense of having required semantics.

Language interfaces should incorporate information on which atoms are predefined, to avoid unnecessary *InternAtom* requests, but these interfaces should make no distinction to their callers between predefined atoms and other atoms. For example, an implementation might maintain a single symbol or atom cache, initializing it with the predefined atoms.

2.4 Input Devices

2.4.1 KEYCODE and KEYSYM

A KEYCODE represents a physical or logical key on an input device. KEYCODE values are in the range from 8 up to and including 255. The numeric value of a KEYCODE provides no intrinsic information to the application. The mapping between keys and KEYCODEs cannot be changed using the X protocol.

A KEYSYM is an encoding of a symbol on the cap of a key. The set of defined KEYSYMs include several common character sets as well as a set of common keyboard symbols. KEYSYMs are specified in Appendix A on page 163.

A list of KEYSYMs is associated with each KEYCODE. The list is intended to convey the set of symbols on the corresponding key. If the list (ignoring trailing *NoSymbol* entries) is a single KEYSYM ''*K*'', then the list is treated as if it were the list ''*K* NoSymbol *K* NoSymbol''. If the list (ignoring trailing *NoSymbol* entries) is a pair of KEYSYMs ''*K1 K2*'', then the list is treated as if it were the list ''*K1 K2 K1 K2*''. If the list (ignoring trailing *NoSymbol* entries) is a triple of KEYSYMs ''*K1 K2 K3*'', then the list is treated as if it were the list ''*K1 K2 K3* NoSymbol''. When an explicit ''void'' element is desired in the list, the value *VoidSymbol* can be used.

The first 4 elements of the list are split into two groups of KEYSYMs. Group 1 contains the first and second KEYSYMs; Group 2 contains the third and fourth KEYSYMs. Within each group, if the second element of the group is *NoSymbol*, then the group should be treated as if the second element were the same as the first element, except when the first element is an alphabetic KEYSYM ''*K*'' for which both lowercase and uppercase forms are defined. In that case, the group should be treated as if the first element were the lowercase form of ''*K*'' and the second element were the uppercase form of ''*K*''.

The rules for obtaining a KEYSYM from a *KeyPress* event make use of only the Group 1 and Group 2 KEYSYMs; no interpretation of other KEYSYMs in the list is defined. The modifier state determines which group to use. Switching between groups is controlled by the KEYSYM named MODE SWITCH, by attaching that KEYSYM to some KEYCODE and attaching that KEYCODE to any one of the modifiers *Mod1* through *Mod5*. This modifier is called the *group modifier*. For any KEYCODE, Group 1 is used when the group modifier is off, and Group 2 is used when the group modifier is on.

2.4.2 Lock Modifier

The *Lock* modifier is interpreted as *CapsLock* when the KEYSYM named CAPS LOCK is attached to some KEYCODE and that KEYCODE is attached to the *Lock* modifier. The *Lock* modifier is interpreted as *ShiftLock* when the KEYSYM named SHIFT LOCK is attached to some KEYCODE and that KEYCODE is attached to the *Lock* modifier. If the *Lock* modifier could be interpreted as both *CapsLock* and *ShiftLock*, the *CapsLock* interpretation is used.

The operation of keypad keys is controlled by the KEYSYM named NUM LOCK, by attaching that KEYSYM to some KEYCODE and attaching that KEYCODE to any one of the modifiers *Mod1* through *Mod5*. This modifier is called the *numlock modifier*. The standard KEYSYMs with the prefix KEYPAD in their name are called *keypad* KEYSYMs; these are KEYSYMs with numeric value in the hexadecimal range #0xFF80 to #0xFFBD inclusive. In addition, implementation-dependent KEYSYMs in the hexadecimal range #0x11000000 to #0x1100FFFF are also keypad KEYSYMs.

Within a group, the choice of KEYSYM is determined by applying the first rule that is satisfied from the following list:

- The *numlock* modifier is on and the second KEYSYM is a keypad KEYSYM. In this case, if the *Shift* modifier is on, or if the *Lock* modifier is on and is interpreted as *ShiftLock*, then the first KEYSYM is used; otherwise, the second KEYSYM is used.
- The *Shift* and *Lock* modifiers are both off. In this case, the first KEYSYM is used.
- The *Shift* modifier is off, and the *Lock* modifier is on and is interpreted as *CapsLock*. In this case, the first KEYSYM is used, but if that KEYSYM is lowercase alphabetic, then the corresponding uppercase KEYSYM is used instead.
- The *Shift* modifier is on, and the *Lock* modifier is on and is interpreted as *CapsLock*. In this case, the second KEYSYM is used, but if that KEYSYM is lowercase alphabetic, then the corresponding uppercase KEYSYM is used instead.
- The *Shift* modifier is on, or the *Lock* modifier is on and is interpreted as *ShiftLock*, or both. In this case, the second KEYSYM is used.

The mapping between KEYCODEs and KEYSYMs is not used directly by the server; it is merely stored for reading and writing by clients.

2.4.3 Pointers

Buttons are always numbered starting with 1.

2.5 Connection Setup

2.5.1 Connection Initiation

The client sends an initial byte of data to identify the byte order to be employed. The value of the byte is #0x42 or #0x6c. The value #0x42 (ASCII uppercase B) means values are transmitted most-significant byte first, and value #0x6c (ASCII lowercase l (ell)) means values are transmitted least-significant byte first. Except where explicitly noted in the protocol, the client transmits all 16-bit and 32-bit quantities with this byte order, and the server returns all 16-bit and 32-bit quantities with this byte order.

Following the byte-order byte, the client sends the following information at connection setup:

protocol-major-version: CARD16
protocol-minor-version: CARD16
authorization-protocol-name: STRING8
authorization-protocol-data: STRING8

The version numbers indicate what version of the protocol the client expects the server to implement.

The authorization name indicates what authorization protocol the client expects the server to use, and the data is specific to that protocol. Valid authorization mechanisms are unspecified. A server that implements a different protocol than the client expects, or that only implements the host-based mechanism, may ignore this information. If both name and data strings are empty, this is to be interpreted as ‘‘no explicit authorization’’.

2.5.2 Server Response

The server sends the client the following information at connection setup:

result: {*Failed*, *Success*, *Authenticate*}

If *result* is *Failed*, the connection is not established and the server sends the client the following additional data:

protocol-major-version: CARD16
protocol-minor-version: CARD16
reason: STRING8

If *result* is *Authenticate*, the server requires further authentication negotiation, and the server sends the client the following additional data:

reason: STRING8

The contents of the reason string are specific to the authorization protocol in use. The semantics of this authentication negotiation are unspecified, except that the negotiation must eventually terminate with a reply from the server containing a *result* of *Failed* or *Success*.

If *result* is *Success*, the connection is established and the server sends the client the following additional data:

protocol-major-version: CARD16
protocol-minor-version: CARD16
vendor: STRING8
release-number: CARD32
resource-id-base, *resource-id-mask*: CARD32
image-byte-order: {*LSBFirst*, *MSBFirst*}
bitmap-scanline-unit: {8, 16, 32}
bitmap-scanline-pad: {8, 16, 32}
bitmap-bit-order: {*LSBitFirst*, *MSBitFirst*}
pixmap-formats: LISTofFORMAT
roots: LISTofSCREEN
motion-buffer-size: CARD32
maximum-request-length: CARD16
min-keycode, *max-keycode*: KEYCODE

where:

FORMAT: [*depth*: CARD8,
bits-per-pixel: {1, 4, 8, 16, 24, 32}
scanline-pad: {8, 16, 32}]

SCREEN: [*root*: WINDOW
width-in-pixels, *height-in-pixels*: CARD16
width-in-millimeters, *height-in-millimeters*: CARD16
allowed-depths: LISTofDEPTH
root-depth: CARD8
root-visual: VISUALID
default-colormap: COLORMAP
white-pixel, *black-pixel*: CARD32
min-installed-maps, *max-installed-maps*: CARD16
backing-store: {`Never`, `WhenMapped`, `Always`}
save-under: BOOL
current-input-masks: SETofEVENT]

DEPTH: [*depth*: CARD8
visuals: LISTofVISUALTYPE]

VISUALTYPE: [*visual-id*: VISUALID
class:
{`StaticGray`, `StaticColor`, `TrueColor`,
`GrayScale`, `PseudoColor`, `DirectColor`}
red-mask, *green-mask*, *blue-mask*: CARD32
bits-per-rgb-value: CARD8
colormap-entries: CARD16]

2.5.3 Server Information

This section describes the information that is global to the server.

The vendor string gives some identification of the owner of the server implementation. The vendor controls the semantics of the release number.

The *resource-id-mask* contains a single contiguous set of bits (at least 18). The client allocates resource IDs for types WINDOW, PIXMAP, CURSOR, FONT, GCONTEXT, and COLORMAP by choosing a value with only some subset of these bits set and ORing it with *resource-id-base*. Only values constructed in this way can be used to name newly created resources over this connection. Resource IDs never have the top three bits set. The client is not restricted to linear or contiguous allocation of resource IDs. Once an ID has been freed, it can be reused, but this should not be necessary. An ID must be unique with respect to the IDs of all other resources, not just other resources of the same type. However, note that the value spaces of resource identifiers, atoms, visual-ids, and keysyms are distinguished by context, and as such, are not required to be disjoint; for example, a given numeric value might be both a valid window ID, a valid atom, and a valid keysym.

The server byte-swaps data to match the client, except that images are always transmitted and received in formats (including byte order) specified by the server. The byte order for images is given by *image-byte-order* and applies to each scanline unit in XY format (bitmap format) and to each pixel value in Z format.

A bitmap is represented in scanline order. Each scanline is padded to a multiple of *bitmap-scanline-pad* bits. The scanline is quantized in multiples of bits as given by *bitmap-scanline-unit*. The *bitmap-scanline-unit* is always less than or equal to the *bitmap-scanline-pad*. Within each unit, the leftmost bit in the bitmap is either the least-significant or most-significant bit in the unit, as given by *bitmap-bit-order*. If a pixmap is represented in XY format, each plane is represented as a bitmap, and the planes appear from most-significant to least-significant in bit order with no padding between planes.

pixmap-formats contains one entry for each depth value. The entry describes the Z format used to represent images of that depth. An entry for a depth is included if any screen supports that depth, and all screens supporting that depth must support only that Z format for that depth. In Z format, the pixels are in scanline order, left to right within a scanline. The number of bits used to hold each pixel is given by *bits-per-pixel*. *bits-per-pixel* may be larger than strictly required by the depth, in which case the least-significant bits are used to hold the pixmap data, and the values of the unused high-order bits are unspecified. When the *bits-per-pixel* is 4, the order of nibbles in the byte is the same as the image byte-order. When the *bits-per-pixel* is 1, the format is identical for bitmap format. Each scanline is padded to a multiple of bits as given by *scanline-pad*. When *bits-per-pixel* is 1, this is identical to *bitmap-scanline-pad*.

The core X protocol supports multiple screens per display. Movement of the pointer between screens is implementation-dependent and is transparent to the protocol. No geometry is defined among screens.

The server may retain the recent history of pointer motion and may do so to a finer granularity than is reported by *MotionNotify* events. The *GetMotionEvents* request makes such history available. The *motion-buffer-size* gives the approximate maximum number of elements in the history buffer.

maximum-request-length specifies the maximum length of a request accepted by the server, in 4-byte units. That is, *length* is the maximum value that can appear in the *length* field of a request. Requests larger than this maximum generate a [LENGTH] error, and the server reads and discards the entire request. The *maximum-request-length* is always at least 4096 units (16384 bytes).

min-keycode and *max-keycode* specify the smallest and largest keycode values transmitted by the server. *min-keycode* is never less than 8, and *max-keycode* is never greater than 255. Not all keycodes in this range are required to have corresponding keys.

2.5.4 Screen Information

This section describes the information that applies per screen.

allowed-depths specifies the pixmap and window depths that are supported. Pixmaps are supported for each depth listed, and windows of that depth are supported if at least one visual type is listed for the depth. A pixmap depth of 1 is always supported and listed, but windows of depth 1 might not be supported. A depth of 0 is never listed, but zero-depth *InputOnly* windows are always supported.

root-depth and *root-visual* specify the depth and visual type of the root window. *width-in-pixels* and *height-in-pixels* specify the size of the root window (which cannot be changed). The class of the root window is always *InputOutput*. *width-in-millimeters* and *height-in-millimeters* can be used to determine the physical size and the aspect ratio.

default-colormap is the colormap initially associated with the root window. Clients with minimal color requirements creating windows of the same depth as the root may want to allocate from this map by default.

black-pixel and *white-pixel* are permanently allocated entries in the default colormap that can be used in implementing a monochrome application. The actual RGB values of *black-pixel* and *white-pixel* are unspecified.

The border of the root window is initially a pixmap filled with *black-pixel*. The initial background of the root window is a pixmap filled with some unspecified two-color pattern using *black-pixel* and *white-pixel*.

min-installed-maps specifies the number of maps that can be guaranteed to be installed simultaneously (with *InstallColormap*), regardless of the number of entries allocated in each map. *max-installed-maps* specifies the maximum number of maps that might possibly be installed simultaneously, depending on their allocations. Multiple static-visual colormaps with identical contents but differing in resource ID should be considered as a single map for the purposes of this number. For the case of a single hardware colormap, both values are 1.[2]

backing-store indicates when the server supports backing store for this screen, although it may be storage-limited in the number of windows it can support at once.

If *save-under* is *True*, the server can support the save-under mode in *CreateWindow* and *ChangeWindowAttributes*, although again it may be storage-limited.

current-input-events is what *GetWindowAttributes* would return for *all-event-masks* on the root window.

2. **Application Usage:** To obtain more accurate information on the colormap and visual capabilities of advanced graphics hardware, applications should use the Extended Visual Information (EVI) Extension.

2.5.5 Visual Information

This section describes the information that applies per visual type.

PseudoColor A pixel value indexes a colormap to produce independent RGB values; the RGB values can be changed dynamically.

GrayScale Treated in the same way as *PseudoColor*, except that the primary that drives the screen is unspecified; thus, the client should always store the same value for red, green, and blue in colormaps.

DirectColor A pixel value is decomposed into separate RGB subfields, and each subfield separately indexes the colormap for the corresponding value. The RGB values can be changed dynamically.

TrueColor Treated in the same way as *DirectColor*, except that the colormap has predefined read-only RGB values. These values are implementation-dependent, but provide linear or near-linear increasing ramps in each primary.

StaticColor Treated in the same way as *PseudoColor*, except that the colormap has predefined read-only RGB values, which are implementation-dependent.

StaticGray Treated in the same way as *StaticColor*, except that the red, green, and blue values are equal for any single pixel value, resulting in shades of gray. *StaticGray* with a two-entry colormap can be thought of as monochrome.

red-mask, *green-mask*, and *blue-mask* are only defined for *DirectColor* and *TrueColor*. Each has one contiguous set of bits set to 1 with no intersections. Usually each mask has the same number of bits set to 1.

bits-per-rgb-value specifies the $\log_2$ of the number of distinct color intensity values (individually) of red, green, and blue. This number need not bear any relation to the number of colormap entries. Actual RGB values are always passed in the protocol within a 16-bit spectrum, with 0 being minimum intensity and 65535 being the maximum intensity. On hardware that provides a linear zero-based intensity ramp, the following relationship exists:

```
hw-intensity = protocol-intensity / (65536 / total-hw-intensities)
```

Colormap entries are indexed from 0. *colormap-entries* defines the number of available colormap entries in a newly created colormap. For *DirectColor* and *TrueColor*, this is usually 2 to the power of the maximum number of bits set to 1 in *red-mask*, *green-mask*, and *blue-mask*.

A server may support a given visual type for more than one depth or for more than one screen. There may be multiple instances of a visual type at a given depth.

2.6 Connection Close

At connection close, all event selections made by the client are discarded. If the client has the pointer actively grabbed, an *UngrabPointer* is performed. If the client has the keyboard actively grabbed, an *UngrabKeyboard* is performed. All passive grabs by the client are released. If the client has the server grabbed, an *UngrabServer* is performed. All selections (see **SetSelectionOwner** on page 77) owned by the client are disowned. If the *mode* parameter of the *SetCloseDownMode* request (see **SetCloseDownMode** on page 74) is *RetainPermanent* or *RetainTemporary*, then all resources (including colormap entries) allocated by the client are marked as permanent or temporary, respectively (but this does not prevent other clients from explicitly destroying them). If *mode* is *Destroy*, all the client's resources are destroyed.

When a client's resources are destroyed, for each window in the client's save-set, if the window is an inferior of a window created by the client, the save-set window is reparented to the closest ancestor such that the save-set window is not an inferior of a window created by the client. If the save-set window is unmapped, a *MapWindow* request is performed on it (even if it was not an inferior of a window created by the client). The reparenting leaves unchanged the absolute coordinates (with respect to the root window) of the upper-left outer corner of the save-set window. After save-set processing, all windows created by the client are destroyed. For each non-window resource created by the client, the appropriate *Free* request is performed. All colors and colormap entries allocated by the client are freed.

A server goes through a cycle of having no connections and having some connections. At every transition to the state of having no connections as a result of a connection closing with a *mode* of *Destroy*, the server resets its state as if it had just been started. This starts by destroying all lingering resources from clients that have terminated with a *mode* of *RetainPermanent* or *RetainTemporary*. It additionally includes deleting all but the predefined atom identifiers, deleting all properties on all root windows, resetting all device maps and attributes (key click, bell volume, acceleration), resetting the access control list, restoring the standard root tiles and cursors, restoring the default font path, and restoring the input focus to state *PointerRoot*.

Closing a connection with a *mode* of *RetainPermanent* or *RetainTemporary* does not reset the server.

2.7 Errors

When a request terminates with an error, the request has no side effects except in the following cases:

- Partial execution may occur if the following requests fail:

  ChangeGC
  ChangeKeyboardControl
  ChangeWindowAttributes
  FreeColors
  PolyText16
  PolyText8
  StoreColors

- An [ALLOC] error can occur at any time in any request.

The following error codes may be returned:

[ACCESS] The client attempted to grab a key/button combination already grabbed by another client.

The client attempted to free a colormap entry not allocated by the client, or to free an entry in a colormap that was created with all entries writable.

The client attempted to store into a read-only or an unallocated colormap entry.

An unauthorized client attempted to modify the access control list.

The client attempted to select an event type that only one client can select at a time when another client has already selected it.

[ALLOC] The server failed to allocate a resource. An [ALLOC] error can occur at any time in any request. Whether partial execution of the request occurs is undefined.

[ATOM] A value for an *ATOM* parameter does not name a defined atom.

[COLORMAP] A value for a *COLORMAP* parameter does not name a defined colormap.

[CURSOR] A value for a *CURSOR* parameter does not name a defined cursor.

[DRAWABLE] A value for a *DRAWABLE* parameter does not name a defined window or pixmap.

[FONT] A value for a *FONT* parameter does not name a defined font.

A value for a *FONTABLE* parameter does not name a defined font or a defined GContext (see **QueryFont** on page 69).

[GCONTEXT] A value for a *GCONTEXT* parameter does not name a defined graphics context.

[IDCHOICE] The value chosen for a resource identifier is not included in the range assigned to the client or is already in use.

[IMPLEMENTATION] The server does not implement some aspect of the request. This error is not listed for any of the requests.

[LENGTH] The length of a request is not the value the server expects.

The length of a request exceeds the maximum length accepted by the server.

[MATCH] An *InputOnly* window is used as a drawable.

In a graphics request, the *GCONTEXT* parameter does not have the same root and depth as the destination *DRAWABLE* parameter.

A parameter (or pair of parameters) has the correct type and range, but it fails to match in some other way required by the request.

[NAME] A font or color of the specified name does not exist.

[PIXMAP] A value for a *PIXMAP* parameter does not name a defined pixmap.

[REQUEST] The major or minor opcode does not specify a valid request.

[VALUE] A numeric value falls outside the set of values accepted by the request.

[WINDOW] A value for a *WINDOW* parameter does not name a defined window.

Note: The [ATOM], [COLORMAP], [CURSOR], [DRAWABLE], [FONT], [GCONTEXT], [PIXMAP], and [WINDOW] errors are also used when the parameter type is extended by union with a set of fixed alternatives; for example, {WINDOW or *PointerRoot* or *None*}.

Additional information returned by each type of error is specified in Section 5.7 on page 158.

2.8 Flow Control and Concurrency

Whenever the server is writing to a given connection, the server may stop reading from that connection; but if the writing would block, it must continue to service other connections. The server is not required to buffer more than a single request per connection at one time. For a given connection to the server, a client can block while reading from the connection but should read when writing would block. Failure on the part of a client to obey this rule may result in a deadlocked connection.[3]

Whether or not a server is implemented with internal concurrency, the overall effect must be as if individual requests are executed to completion in some serial order, and requests from a given connection must be executed in delivery order (that is, the total execution order is a shuffle of the individual streams). The execution of a request includes validating all parameters, collecting all data for any reply, and generating and queueng all required events. However, it does not include the actual transmission of the reply and the events. In addition, the effect of any other cause that can generate multiple events (for example, activation of a grab or pointer motion) must effectively generate and queue all required events indivisibly with respect to all other causes and requests. For a request from a given client, any events destined for that client that are caused by executing the request must be sent to the client before any reply or error is sent.

3. **Application Usage:** Deadlock is especially likely if the transport layer has little buffering or if the client sends many requests without ever reading replies or checking for errors and events.

Chapter 3

Requests

This chapter lists alphabetically each type of request that a client can send to a server.

This chapter follows the syntactic conventions set out in Section 1.3 on page 3.

AllocColor

cmap: COLORMAP
red, *green*, *blue*: CARD16
→
pixel: CARD32
red, *green*, *blue*: CARD16

Errors: [ALLOC], [COLORMAP]

This request allocates a read-only colormap entry corresponding to the closest RGB values provided by the hardware. It also returns the pixel and the RGB values actually used. Multiple clients requesting the same effective RGB values can be assigned the same read-only entry, allowing entries to be shared.

AllocColorCells

cmap: COLORMAP
colors, *planes*: CARD16
contiguous: BOOL
→
pixels, *masks*: LISTofCARD32

Errors: [ALLOC], [COLORMAP], [VALUE]

colors must be positive and *planes* must be non-negative, or a [VALUE] error results. If C colors and P planes are requested, then C pixels and P masks are returned. No mask has any bits in common with any other mask or with any of the pixels. By ORing together masks and pixels, $C * 2^P$ distinct pixels can be produced; all are allocated writable by the request. For *GrayScale* or *PseudoColor*, each mask has exactly one bit set to 1; for *DirectColor*, each mask has exactly three bits set to 1. If *contiguous* is *True* and if all masks are ORed together, a single contiguous set of bits is formed for *GrayScale* or *PseudoColor*, and three contiguous sets of bits (one within each pixel subfield) for *DirectColor*. The RGB values of the allocated entries are unspecified.

AllocColorPlanes

cmap: COLORMAP
colors, *reds*, *greens*, *blues*: CARD16
contiguous: BOOL
→
pixels: LISTofCARD32
red-mask, *green-mask*, *blue-mask*: CARD32

Errors: [ALLOC], [COLORMAP], [VALUE]

colors must be positive and *reds*, *greens*, and *blues* must be non-negative, or a [VALUE] error results. If C colors, R reds, G greens, and B blues are requested, then C pixels are returned, and the masks have R, G, and B bits set, respectively. If *contiguous* is *True*, then each mask has a contiguous set of bits. No mask has any bits in common with any other mask or with any of the pixels. For *DirectColor*, each mask lies within the corresponding pixel subfield. By ORing together subsets of masks with pixels, $C * 2^{R+G+B}$ distinct pixels can be produced; all of these are allocated writable by the request. The initial RGB values of the allocated entries are unspecified. In the colormap, there are only $C * 2^{R}$ independent red entries, $C * 2^{G}$ independent green entries, and $C * 2^{B}$ independent blue entries. This is true even for *PseudoColor*. When the colormap entry for a pixel value is changed using *StoreColors* or *StoreNamedColor*, the pixel is decomposed according to the masks and the corresponding independent entries are updated.

AllocNamedColor

cmap: COLORMAP
name: STRING8
→
pixel: CARD32
exact-red, *exact-green*, *exact-blue*: CARD16
visual-red, *visual-green*, *visual-blue*: CARD16

Errors: [ALLOC], [COLORMAP], [NAME]

This request looks up the named color with respect to the screen associated with *cmap*. Then it does an *AllocColor* on *cmap*. The exact RGB values specify the true values for the color, and the visual values specify the values actually used in the colormap. *name* is compared, case-insensitively, against valid color names. If *name* does not use the ISO Latin-1 encoding, the results are implementation-dependent.

AllowEvents

mode: {*AsyncPointer*, *SyncPointer*, *ReplayPointer*, *AsyncKeyboard*, *SyncKeyboard*, *ReplayKeyboard*, *AsyncBoth*, *SyncBoth*}
time: TIMESTAMP or *CurrentTime*

Errors: [VALUE]

This request releases some queued events if the client has caused a device to freeze. The request has no effect if *time* is earlier than the last-grab time of the most recent active grab for the client or later than the current server time.

The following effects are specific to values of *mode*:

AsyncPointer If the pointer is frozen by the client, pointer event processing continues normally. If the pointer is frozen twice by the client on behalf of two separate grabs, the request thaws for both. The request has no effect if the pointer is not frozen by the client, but the pointer need not be grabbed by

the client.

SyncPointer If the pointer is frozen and actively grabbed by the client, pointer event processing continues normally until the next *ButtonPress* or *ButtonRelease* event is reported to the client, at which time the pointer again appears to freeze. However, if the reported event causes the pointer grab to be released, then the pointer does not freeze. The request has no effect if the pointer is not frozen by the client or if the pointer is not grabbed by the client.

ReplayPointer If the pointer is actively grabbed by the client and is frozen as the result of an event having been sent to the client (either from the activation of a *GrabButton* or from a previous *AllowEvents* with mode *SyncPointer* but not from a *GrabPointer*), then the pointer grab is released and that event is completely reprocessed, this time ignoring any passive grabs at or above (towards the root) the grab-window of the grab just released. The request has no effect if the pointer is not grabbed by the client or if the pointer is not frozen as the result of an event.

AsyncKeyboard If the keyboard is frozen by the client, keyboard event processing continues normally. If the keyboard is frozen twice by the client on behalf of two separate grabs, the request thaws for both. The request has no effect if the keyboard is not frozen by the client, but the keyboard need not be grabbed by the client.

SyncKeyboard If the keyboard is frozen and actively grabbed by the client, keyboard event processing continues normally until the next *KeyPress* or *KeyRelease* event is reported to the client, at which time the keyboard again appears to freeze. However, if the reported event causes the keyboard grab to be released, then the keyboard does not freeze. The request has no effect if the keyboard is not frozen by the client or if the keyboard is not grabbed by the client.

ReplayKeyboard If the keyboard is actively grabbed by the client and is frozen as the result of an event having been sent to the client (either from the activation of a *GrabKey* or from a previous *AllowEvents* with mode *SyncKeyboard* but not from a *GrabKeyboard*), then the keyboard grab is released and that event is completely reprocessed, this time ignoring any passive grabs at or above (towards the root) the grab-window of the grab just released. The request has no effect if the keyboard is not grabbed by the client or if the keyboard is not frozen as the result of an event.

SyncBoth If both pointer and keyboard are frozen by the client, event processing (for both devices) continues normally until the next *ButtonPress*, *ButtonRelease*, *KeyPress*, or *KeyRelease* event is reported to the client for a grabbed device (button event for the pointer, key event for the keyboard), at which time the devices again appear to freeze. However, if the reported event causes the grab to be released, then the devices do not freeze (but if the other device is still grabbed, then a subsequent event for it still causes both devices to freeze). The request has no effect unless both pointer and keyboard are frozen by the client. If the pointer or keyboard is frozen twice by the client on behalf of two separate grabs, the request thaws for both (but a subsequent freeze for *SyncBoth* only freezes each device once).

AsyncBoth If the pointer and the keyboard are frozen by the client, event processing for both devices continues normally. If a device is frozen twice by the

client on behalf of two separate grabs, the request thaws for both. The request has no effect unless both pointer and keyboard are frozen by the client.

AsyncPointer, *SyncPointer*, and *ReplayPointer* have no effect on processing of keyboard events. *AsyncKeyboard*, *SyncKeyboard*, and *ReplayKeyboard* have no effect on processing of pointer events.

It is possible for both a pointer grab and a keyboard grab to be active simultaneously (by the same or different clients). When a device is frozen on behalf of either grab, no event processing is performed for the device. It is possible for a single device to be frozen because of both grabs. In this case, the freeze must be released on behalf of both grabs before events can again be processed. If a device is frozen twice by a single client, then a single *AllowEvents* releases both.

Bell

percent: INT8

Errors: [VALUE]

This request rings the bell at a volume relative to the base volume, if possible. *percent* can range from –100 to 100 inclusive, or a [VALUE] error results.

Values from –100 up to and including 0 specify a volume that varies linearly from silence (–100) to the base volume (0):

base + [(*base* * *percent*) / 100]

Values from 0 up to and including 100 specify a volume that varies linearly from the base volume (0) up to the maximum volume (100):

base – [(*base* * *percent*) / 100] + *percent*

ChangeActivePointerGrab

event-mask: SETofPOINTEREVENT
cursor: CURSOR or *None*
time: TIMESTAMP or *CurrentTime*

Errors: [CURSOR], [VALUE]

This request changes the specified dynamic parameters if the pointer is actively grabbed by the client and *time* is no earlier than the *last-pointer-grab* time and no later than the current server time. The interpretation of *event-mask* and *cursor* are the same as in *GrabPointer*. This request has no effect on the parameters of any passive grabs established with *GrabButton*.

ChangeGC

gc: GCONTEXT
value-mask: BITMASK
value-list: LISTofVALUE

Errors: [ALLOC], [FONT], [GCONTEXT], [MATCH], [PIXMAP], [VALUE]

This request changes components in *gc*. *value-mask* and *value-list* specify which components are to be changed. The values and restrictions are the same as for *CreateGC*.

Changing the *clip-mask* also overrides any previous *SetClipRectangles* request on the context. Changing *dash-offset* or *dashes* overrides any previous *SetDashes* request on the context.

The order in which components are verified and altered is implementation-dependent. If an error is generated, a subset of the components may have been altered.

ChangeHosts

mode: {*Insert*, *Delete*}
host: HOST

Errors: [ACCESS], [VALUE]

This request adds *host* to, or removes it from, the access control list. When the access control mechanism is enabled and a host not in the list tries to establish a connection to the server, the server refuses the connection.

The client must have been granted permission by a implementation-dependent method to execute this request, or an [ACCESS] error results.

The following address families are defined:

- For the Internet family, IPv4 addresses are 4 bytes long. The address bytes are in standard IP order.
- For the Chaos family, the address is 2 bytes long.

A server is not required to support these families and may support families not listed here. Use of an unsupported family, an improper address format, or an improper address length within a supported family results in a [VALUE] error.

ChangeKeyboardControl

value-mask: BITMASK
value-list: LISTofVALUE

Errors: [MATCH], [VALUE]

This request controls various aspects of the keyboard. *value-mask* and *value-list* specify which controls are to be changed. The possible values are:

Control	Type
key-click-percent	INT8
bell-percent	INT8
bell-pitch	INT16
bell-duration	INT16
led	CARD8
led-mode	{*On*, *Off*}
key	KEYCODE
auto-repeat-mode	{*On*, *Off*, *Default*}

key-click-percent sets the volume for key clicks between 0 (off) and 100 (loud) inclusive, if possible. Setting to −1 restores the implementation-dependent default. Other negative values generate a [VALUE] error.

bell-percent sets the base volume for the bell between 0 (off) and 100 (loud) inclusive, if possible. Setting to −1 restores the implementation-dependent default. Other negative values generate a [VALUE] error.

bell-pitch sets the pitch (specified in Hz) of the bell, if possible. Setting to −1 restores the implementation-dependent default. Other negative values generate a [VALUE] error.

bell-duration sets the duration of the bell (specified in milliseconds), if possible. Setting to –1 restores the implementation-dependent default. Other negative values generate a [VALUE] error.

If both *led-mode* and *led* are specified, then the state of that LED is changed, if possible. If only *led-mode* is specified, then the state of all LEDs is changed, if possible. At most 32 LEDs, numbered from one, are supported. No standard interpretation of LEDs is defined. It is a [MATCH] error if *led* is specified without *led-mode*.

If both *auto-repeat-mode* and *key* are specified, then the auto-repeat mode of that key is changed, if possible. If only *auto-repeat-mode* is specified, then the global auto-repeat mode for the entire keyboard is changed, if possible, without affecting the per-key settings. It is a [MATCH] error if *key* is specified without *auto-repeat-mode*. Each key has an individual mode of whether or not it should auto-repeat and a default setting for that mode. In addition, there is a global mode of whether auto-repeat should be enabled or not and a default setting for that mode. When the global mode is *On*, keys should obey their individual auto-repeat modes. When the global mode is *Off*, no keys should auto-repeat. An auto-repeating key generates alternating *KeyPress* and *KeyRelease* events. When a key is used as a modifier, it is desirable for the key not to auto-repeat, regardless of the auto-repeat setting for that key.

A bell generator connected with the console but not directly on the keyboard is treated as if it were part of the keyboard.

The order in which controls are verified and altered is implementation-dependent. If an error is generated, a subset of the controls may have been altered.

ChangeKeyboardMapping

first-keycode: KEYCODE
keysyms-per-keycode: CARD8
keysyms: LISTofKEYSYM

Errors: [ALLOC], [VALUE]

This request defines the symbols for the specified number of keycodes, starting with *first-keycode*. The symbols for keycodes outside this range remained unchanged. The number of elements in the keysyms list must be a multiple of *keysyms-per-keycode*, or a [LENGTH] error results. *first-keycode* must be greater than or equal to *min-keycode* as returned in the connection setup, or a [VALUE] error results; and:

first-keycode + (*keysyms-length* / *keysyms-per-keycode*) –1

must be less than or equal to *max-keycode* as returned in the connection setup, or a [VALUE] error results. KEYSYM number N (counting from 0) for keycode K has an index (counting from 0) of:

(K – *first-keycode*) * *keysyms-per-keycode* + N

in keysyms. The client can arbitrarily choose *keysyms-per-keycode* to be large enough to hold all desired symbols. A special KEYSYM value of *NoSymbol* should be used to fill in unused elements for individual keycodes. It is valid for *NoSymbol* to appear in non-trailing positions of the effective list for a keycode.

This request generates a *MappingNotify* event.

There is no requirement that the server interpret this mapping; it is merely stored for reading and writing by clients (see Section 2.4).

ChangePointerControl

do-acceleration, *do-threshold*: BOOL
acceleration-numerator, *acceleration-denominator*: INT16
threshold: INT16

Errors: [VALUE]

This request defines how the pointer moves. The acceleration is a multiplier for movement expressed as a fraction. For example, specifying 3/1 means the pointer moves three times as fast as normal. The fraction can be rounded arbitrarily by the server. Acceleration only takes effect if the pointer moves more than threshold number of pixels at once and only applies to the amount beyond the threshold. Setting a value to –1 restores the default. Other negative values generate a *Value* error, as does a 0 value for *acceleration-denominator*.

ChangeProperty

window: WINDOW
property, *type*: ATOM
format: {8, 16, 32}
mode: {*Replace*, *Prepend*, *Append*}
data: LISTofINT8 or LISTofINT16 or LISTofINT32

Errors: [ALLOC], [ATOM], [MATCH], [VALUE], [WINDOW]

This request alters *property* of *window*. *type* is uninterpreted by the server. *format* specifies whether the data should be viewed as a list of 8-bit, 16-bit, or 32-bit quantities so that the server can correctly byte-swap as necessary.

If *mode* is *Replace*, the previous property value is discarded. If *mode* is *Prepend* or *Append*, then *type* and *format* must match the existing property value, or a [MATCH] error results. If the property is not defined, it is treated as defined with the correct *type* and *format* with zero-length data. For *Prepend*, the data is prepended to any existing data, and for *Append*, it is appended to any existing data.

This request generates a *PropertyNotify* event on the window.

The lifetime of a property is not tied to the storing client. Properties remain until explicitly deleted, until the window is destroyed, or until server reset (see Section 2.6).

The maximum size of a property is implementation-dependent and may vary dynamically.

ChangeSaveSet

window: WINDOW
mode: {*Insert*, *Delete*}

Errors: [MATCH], [VALUE], [WINDOW]

This request adds *window* to, or removes it from, the client's save-set (see Section 2.6). *window* must have been created by some other client, or a [MATCH] error results.

When windows are destroyed, the server automatically removes them from the save-set.

ChangeWindowAttributes

window: WINDOW
value-mask: BITMASK
value-list: LISTofVALUE

Errors: [ACCESS], [COLORMAP], [CURSOR], [MATCH], [PIXMAP], [VALUE], [WINDOW]

value-mask and *value-list* specify the attributes to change. The values and restrictions are the same as for *CreateWindow*.

Setting a new background, whether by *background-pixmap* or *background-pixel*, overrides any previous background. Setting a new border, whether by *border-pixel* or *border-pixmap*, overrides any previous border.

Changing the background does not cause the window contents to be changed. Setting the border or changing the background such that the border tile origin changes causes the border to be repainted. Changing the background of a root window to *None* or *ParentRelative* restores the default background pixmap. Changing the border of a root window to *CopyFromParent* restores the default border pixmap.

Changing the *win-gravity* does not affect the current position of the window.

Changing the *backing-store* of an obscured window to *WhenMapped* or *Always* or changing the *backing-planes*, *backing-pixel*, or *save-under* of a mapped window may have no immediate effect.

Multiple clients can select input on the same window; their *event-mask*s are disjoint. When an event is generated, it is reported to all interested clients. However, only one client at a time can select for *SubstructureRedirect*, only one client at a time can select for *ResizeRedirect*, and only one client at a time can select for *ButtonPress*. Violating these restrictions produces an [ACCESS] error.

There is only one *do-not-propagate-mask* for a window, not one per client.

Changing the colormap of a window (by defining a new map, not by changing the contents of the existing map) generates a *ColormapNotify* event. It is unspecified whether changing the colormap of a visible window has an immediate effect on the screen (see *InstallColormap* request).

Changing the cursor of a root window to *None* restores the default cursor.

The order in which attributes are verified and altered is unspecified. If an error is generated, a subset of the attributes may have been altered.

CirculateWindow

window: WINDOW
direction: {*RaiseLowest*, *LowerHighest*}

Errors: [VALUE], [WINDOW]

If some other client has selected *SubstructureRedirect* on *window*, then a *CirculateRequest* event is generated, and no further processing is performed. Otherwise, the following is performed, and then a *CirculateNotify* event is generated if *window* is actually restacked.

For *RaiseLowest*, *CirculateWindow* raises the lowest mapped child (if any) that is occluded by another child to the top of the stack. For *LowerHighest*, *CirculateWindow* lowers the highest mapped child (if any) that occludes another child to the bottom of the stack. Exposure processing is performed on formerly obscured windows.

ClearArea

window: WINDOW
x, *y*: INT16
width, *height*: CARD16
exposures: BOOL

Errors: [MATCH], [VALUE], [WINDOW]

The *x* and *y* coordinates are relative to the window's origin and specify the upper-left corner of the rectangle. If *width* is 0, it is replaced with the current width of the window minus *x*. If *height* is 0, it is replaced with the current height of the window minus *y*. If the window has a defined background tile, the rectangle is tiled with a *plane-mask* of all ones and function of *Copy* and a *subwindow-mode* of *ClipByChildren*. If the window has background *None*, the contents of the window are not changed. In either case, if *exposures* is *True*, then one or more exposure events are generated for regions of the rectangle that are either visible or are being retained in a backing store.

It is a [MATCH] error to use an *InputOnly* window in this request.

CloseFont

font: FONT

Errors: [FONT]

This request deletes the association between the resource ID and *font*. The font itself is freed when no other resource references it.

ConfigureWindow

window: WINDOW
value-mask: BITMASK
value-list: LISTofVALUE

Errors: [MATCH], [VALUE], [WINDOW]

This request changes the configuration of *window*. *value-mask* and *value-list* specify which values are to be given. The possible values are:

Attribute	Type
x	INT16
y	INT16
width	CARD16
height	CARD16
border-width	CARD16
sibling	WINDOW
stack-mode	{*Above*, *Below*, *TopIf*, *BottomIf*, *Opposite*}

The *x* and *y* coordinates are relative to the parent's origin and specify the position of the upper-left outer corner of the window. *width* and *height* specify the inside size, not including the border, and must be nonzero, or a [VALUE] error results. Those values not specified are taken from the existing geometry of the window. Changing just *border-width* leaves the outer-left corner of the window in a fixed position but moves the absolute position of the window's origin. It is a [MATCH] error to specify a nonzero *border-width* for an *InputOnly* window.

If *override-redirect* of the window is *False* and some other client has selected *SubstructureRedirect* on the parent, a *ConfigureRequest* event is generated, and no further processing is performed.

If some other client has selected *ResizeRedirect* on the window and the inside width or height of the window is being changed, a *ResizeRequest* event is generated, and the current inside width and height are used instead. The *override-redirect* attribute of the window has no effect on *ResizeRedirect* and *SubstructureRedirect* on the parent has precedence over *ResizeRedirect* on the window.

The geometry of the window is changed as specified, the window is restacked among siblings, and a *ConfigureNotify* event is generated if the state of the window actually changes. If the inside width or height of the window has actually changed, then children of the window are affected, according to their window gravity. Exposure processing is performed on formerly obscured windows (including the window itself and its inferiors if regions of them were obscured but now are not). Exposure processing is also performed on any new regions of the window (as a result of increasing the width or height) and on any regions where window contents are lost.

If the inside width or height of a window is not changed but the window is moved or its border is changed, then the contents of the window are not lost but move with the window. Changing the inside width or height of the window causes its contents to be moved or lost, depending on the bit-gravity of the window. It also causes children to be reconfigured, depending on their window gravity. For a change of width and height of *W* and *H*, we define the [*x*, *y*] pairs as:

Direction	Deltas
NorthWest	[0, 0]
North	[*W*/2, 0]
NorthEast	[*W*, 0]
West	[0, *H*/2]
Center	[*W*/2, *H*/2]
East	[*W*, *H*/2]
SouthWest	[0, *H*]
South	[*W*/2, *H*]
SouthEast	[*W*, *H*]

When a window with one of these bit-gravities is resized, the corresponding pair defines the change in position of each pixel in the window. When a window with one of these win-gravities has its parent window resized, the corresponding pair defines the change in position of the window within the parent. This repositioning generates a *GravityNotify* event. *GravityNotify* events are generated after the *ConfigureNotify* event is generated.

A gravity of *Static* indicates that the contents or origin should not move relative to the origin of the root window. If the change in size of the window is coupled with a change in position of [*X*, *Y*], then for *bit-gravity* the change in position of each pixel is [−*X*, −*Y*] and for *win-gravity* the change in position of a child when its parent is so resized is [−*X*, −*Y*]. *Static* gravity still only takes effect when the width or height of the window is changed, not when the window is simply moved.

A *bit-gravity* of *Forget* indicates that the window contents are always discarded after a size change, even if *backing-store* or *save-under* has been requested. The window is tiled with its background (except, if no background is defined, the existing screen contents are not altered) and 0 or more exposure events are generated.

The contents and borders of inferiors are not affected by their parent's *bit-gravity*. A server may ignore *bit-gravity* and use *Forget* instead.

A *win-gravity* of *Unmap* is like *NorthWest*, but the child is also unmapped when the parent is resized, and an *UnmapNotify* event is generated. *UnmapNotify* events are generated after the *ConfigureNotify* event is generated.

If a *sibling* and a *stack-mode* are specified, the window is restacked as follows:

Above The window is placed just above the sibling.

Below The window is placed just below the sibling.

TopIf If the sibling occludes the window, then the window is placed at the top of the stack.

BottomIf If the window occludes the sibling, then the window is placed at the bottom of the stack.

Opposite If the sibling occludes the window, then the window is placed at the top of the stack. Otherwise, if the window occludes the sibling, then the window is placed at the bottom of the stack.

If a *stack-mode* is specified but no *sibling* is specified, the window is restacked as follows:

Above The window is placed at the top of the stack.

Below The window is placed at the bottom of the stack.

TopIf If any sibling occludes the window, then the window is placed at the top of the stack.

BottomIf If the window occludes any sibling, then the window is placed at the bottom of the stack.

Opposite If any sibling occludes the window, then the window is placed at the top of the stack. Otherwise, if the window occludes any sibling, then the window is placed at the bottom of the stack.

It is a [MATCH] error if a *sibling* is specified without a *stack-mode* or if the window is not actually a sibling.

The computations for *BottomIf*, *TopIf*, and *Opposite* are performed with respect to the window's final geometry (as controlled by the other parameters to the request), not to its initial geometry.

Attempts to configure a root window have no effect.

ConvertSelection

selection, *target*: ATOM
property: ATOM or *None*
requestor: WINDOW
time: TIMESTAMP or *CurrentTime*

Errors: [ATOM], [WINDOW]

If *selection* has an owner, the server sends a *SelectionRequest* event to that owner. If no owner for *selection* exists, the server generates a *SelectionNotify* event to the requestor with property *None*. The parameters are passed on unchanged in either of the events.

CopyArea

src-drawable, *dst-drawable*: DRAWABLE
gc: GCONTEXT
src-x, *src-y*: INT16
width, *height*: CARD16
dst-x, *dst-y*: INT16

Errors: [DRAWABLE], [GCONTEXT], [MATCH]

This request combines the specified rectangle of *src-drawable* with the specified rectangle of *dst-drawable*. The *src-x* and *src-y* coordinates are relative to the origin of *src-drawable*. *dst-x* and *dst-y* are relative to the origin of *dst-drawable*, each pair specifying the upper-left corner of the rectangle. *src-drawable* must have the same root and the same depth as *dst-drawable*, or a [MATCH] error results.

If regions of the source rectangle are obscured and have not been retained in backing store or if regions outside the boundaries of the source drawable are specified, then those regions are not copied, but the following occurs on all corresponding destination regions that are either visible or are retained in backing-store. If *dst-drawable* is a window with a background other than *None*, these corresponding destination regions are tiled (with *plane-mask* of all ones and function *Copy*) with that background. Regardless of tiling and whether the destination is a window or a pixmap, if *graphics-exposures* in *gc* is *True*, then *GraphicsExpose* events for all corresponding destination regions are generated.

If *graphics-exposures* is *True* but no *GraphicsExpose* events are generated, then a *NoExpose* event is generated.

The following components of *gc* must be provided: *function*, *plane-mask*, *subwindow-mode*, *graphics-exposures*, *clip-x-origin*, *clip-y-origin*, *clip-mask*.

CopyColormapAndFree

mid, *src-cmap*: COLORMAP

Errors: [ALLOC], [COLORMAP], [IDCHOICE]

This request creates a colormap of the same visual type and for the same screen as *src-cmap*, and it associates identifier *mid* with it. It also moves all of the client's existing allocations from *src-cmap* to the new colormap with their color values intact and their read-only or writable characteristics intact, and it frees those entries in *src-cmap*. Color values in other entries in the new colormap are unspecified. If *src-cmap* was created by the client with *alloc=All* (see *CreateColormap* request), then the new colormap is also created with *alloc=All*, all color values for all entries are copied from *src-cmap*, and then all entries in *src-cmap* are freed. If *src-cmap* was not created by the client with *alloc=All*, then the allocations to be moved are all those pixels and planes that have been allocated by the client using either *AllocColor*, *AllocNamedColor*, *AllocColorCells*, or *AllocColorPlanes*, and that have not been freed since they were allocated.

CopyGC

src-gc, *dst-gc*: GCONTEXT
value-mask: BITMASK

Errors: [ALLOC], [GCONTEXT], [MATCH], [VALUE]

This request copies components from *src-gc* to *dst-gc*. The *value-mask* specifies which components to copy, as for *CreateGC*. The two graphics contexts must have the same root and the same depth, or a [MATCH] error results.

CopyPlane

src-drawable, *dst-drawable*: DRAWABLE
gc: GCONTEXT
src-x, *src-y*: INT16
width, *height*: CARD16
dst-x, *dst-y*: INT16
bit-plane: CARD32

Errors: [DRAWABLE], [GCONTEXT], [MATCH], [VALUE]

src-drawable must have the same root as *dst-drawable*, or a [MATCH] error results, but it need not have the same depth.

bit-plane must have exactly one bit set to 1 and the value of *bit-plane* must be less than 2^N, where N is the depth of *src-drawable*, or a [VALUE] error results.

Effectively, a pixmap of the same depth as *dst-drawable* and with size specified by the source region is formed using the foreground/background pixels in *gc* (*foreground* everywhere *bit-plane* in *src-drawable* contains a bit set to 1, *background* everywhere *bit-plane* contains a bit set to 0), and the equivalent of a *CopyArea* is performed, with all the same exposure semantics. This can also be thought of as using the specified region of the source *bit-plane* as a stipple with a fill-style of *OpaqueStippled* for filling a rectangular area of the destination.

The following components of *gc* must be provided: *function*, *plane-mask*, *foreground*, *background*, *subwindow-mode*, *graphics-exposures*, *clip-x-origin*, *clip-y-origin*, *clip-mask*.

CreateColormap

mid: COLORMAP
visual: VISUALID
window: WINDOW
alloc: {*None*, *All*}

Errors: [ALLOC], [IDCHOICE], [MATCH], [VALUE], [WINDOW]

This request creates a colormap of type *visual* for the screen on which the window resides and associates the identifier *mid* with it. *visual* must be supported by the screen, or a [MATCH] error results. The initial values of the colormap entries are unspecified for classes *GrayScale*, *PseudoColor*, and *DirectColor*. For *StaticGray*, *StaticColor*, and *TrueColor*, the entries have defined values, but those values are specific to the visual and are not defined by the core X protocol. For *StaticGray*, *StaticColor*, and *TrueColor*, *alloc* must be specified as *None*, or a [MATCH] error results. For the other classes, if *alloc* is *None*, the colormap initially has no allocated entries, and clients can allocate entries.

If *alloc* is *All*, then the entire colormap is writable. The initial values of all allocated entries are unspecified. The [ALLOC] error may result if the server cannot allocate a colormap with all cells writable. For *GrayScale* and *PseudoColor*, the effect is as if an *AllocColorCells* request returned all pixel values from 0 to N –1, where N is the *colormap-entries* value in *visual*. For *DirectColor*, the effect is as if an *AllocColorPlanes* request returned a pixel value of 0 and *red-mask*, *green-mask*, and *blue-mask* values containing the same bits as the corresponding masks in *visual*. However, these values cannot be freed with *FreeColors*.

CreateCursor

cid: CURSOR
source: PIXMAP
mask: PIXMAP or *None*
fore-red, *fore-green*, *fore-blue*: CARD16
back-red, *back-green*, *back-blue*: CARD16
x, *y*: CARD16

Errors: [ALLOC], [IDCHOICE], [MATCH], [PIXMAP]

This request creates a cursor and associates identifier *cid* with it. The foreground and background RGB values must be specified, even if the server only has a *StaticGray* or *GrayScale* screen. The foreground is used for the bits set to 1 in the source, and the background is used for the bits set to 0. Both source and mask (if specified) must have depth one, or a [MATCH] error results, but they can have any root. The *mask* pixmap defines the shape of the cursor. That is, the bits set to 1 in *mask* define which source pixels are displayed, and where *mask* has bits set to 0, the corresponding bits of the source pixmap are ignored. If no *mask* is given, all pixels of the source are displayed. Any *mask* must be the same size as the source, or a [MATCH] error results. The *x* and *y* coordinates define the hotspot relative to the source's origin and must be a point within the source, or a [MATCH] error results.

The components of the cursor may be transformed in implementation-dependent ways to meet display limitations.

The effect on the cursor of subsequent drawing in the source or mask pixmap is unspecified. It is unspecified whether the server makes a copy of the pixmap.

CreateGC

cid: GCONTEXT
drawable: DRAWABLE
value-mask: BITMASK
value-list: LISTofVALUE

Errors: [ALLOC], [DRAWABLE], [FONT], [IDCHOICE], [MATCH], [PIXMAP], [VALUE]

This request creates a graphics context and assigns the identifier *cid* to it. The graphics context can be used with any destination drawable having the same root and depth as *drawable*; use with other drawables results in a [MATCH] error.

value-mask and *value-list* specify which components to explicitly initialize. The context components are:

Component	Type
function	{*Clear*, *And*, *AndReverse*, *Copy*, *AndInverted*, *NoOp*, *Xor*, *Or*, *Nor*, *Equiv*, *Invert*, *OrReverse*, *CopyInverted*, *OrInverted*, *Nand*, *Set*}
plane-mask	CARD32
foreground	CARD32
background	CARD32
line-width	CARD16
line-style	{*Solid*, *OnOffDash*, *DoubleDash*}

Component	Type
cap-style	{*NotLast*, *Butt*, *Round*, *Projecting*}
join-style	{*Miter*, *Round*, *Bevel*}
fill-style	{*Solid*, *Tiled*, *OpaqueStippled*, *Stippled*}
fill-rule	{*EvenOdd*, *Winding*}
arc-mode	{*Chord*, *PieSlice*}
tile	PIXMAP
stipple	PIXMAP
tile-stipple-x-origin	INT16
tile-stipple-y-origin	INT16
font	FONT
subwindow-mode	{*ClipByChildren*, *IncludeInferiors*}
graphics-exposures	BOOL
clip-x-origin	INT16
clip-y-origin	INT16
clip-mask	PIXMAP or None
dash-offset	CARD16
dashes	CARD8

In graphics operations, given a source pixel *src* and destination pixel *dst*, the result is computed bitwise on corresponding bits of the pixels; that is, a Boolean operation specified by *function* is performed in each bit plane. *plane-mask* restricts the operation to a subset of planes, so the result is:

((*src function dst*) AND *plane-mask*) OR (*dst* AND (NOT *plane-mask*))

The values of *foreground*, *background*, and *plane-mask* are not range-checked but are truncated to the appropriate number of bits.

The meanings of the functions are:

Function	Operation
Clear	0
And	*src* AND *dst*
AndReverse	*src* AND (NOT *dst*)
Copy	*src*
AndInverted	(NOT *src*) AND *dst*
NoOp	*dst*
Xor	*src* XOR *dst*
Or	*src* OR *dst*
Nor	(NOT *src*) AND (NOT *dst*)
Equiv	(NOT *src*) XOR *dst*
Invert	NOT *dst*
OrReverse	*src* OR (NOT *dst*)
CopyInverted	NOT *src*
OrInverted	(NOT *src*) OR *dst*
Nand	(NOT *src*) OR (NOT *dst*)
Set	1

line-width is measured in pixels and can be greater than or equal to one, a wide line, or the special value 0, a thin line.

Wide lines are drawn centered on the path described by the graphics request. Unless otherwise specified by the join or cap style, the bounding box of a wide line with endpoints [*x1*, *y1*], [*x2*, *y2*]

and width *w* is a rectangle with vertices at the following real coordinates:

$[x1-w*sn/2), y1+(w*cs/2)], [x1+(w*sn/2), y1-w*cs/2)],$
$[x2-w*sn/2), y2+(w*cs/2)], [x2+(w*sn/2), y2-w*cs/2)]$

sn is the sine of the angle of the line and *cs* is the cosine of the angle of the line. A pixel is part of the line (and hence drawn) if the center of the pixel is fully inside the bounding box, which is viewed as having infinitely thin edges. If the center of the pixel is exactly on the bounding box, it is part of the line if and only if the interior is immediately to its right (x increasing direction). Pixels with centers on a horizontal edge are a special case and are part of the line if and only if the interior or the boundary is immediately below (y increasing direction) and if the interior or the boundary is immediately to the right (x increasing direction).[4]

Thin lines (0 line-width) are one-pixel-wide lines drawn using an unspecified algorithm. There are only two constraints on this algorithm. First, if a line is drawn unclipped from [*x1*, *y1*] to [*x2*, *y2*] and another line is drawn unclipped from [*x1+dx*, *y1+dy*] to [*x2+dx*, *y2+dy*], then a point [*x*, *y*] is touched by drawing the first line if and only if the point [*x+dx*, *y+dy*] is touched by drawing the second line. Second, the effective set of points comprising a line cannot be affected by clipping. Thus, a point is touched in a clipped line if and only if the point lies inside the clipping region and the point would be touched by the line when drawn unclipped.

The endpoints of a wide line with a *line-style* of *Solid* can be specified in either order without affecting the pixels drawn (unless a certain order is required to join with subsequent lines).

A *line-width* of 0 may differ from a *line-width* of 1 in which pixels are drawn. To obtain precise and uniform results across all displays, a client should always use a *line-width* of 1, rather than a *line-width* of 0.

line-style defines which sections of a line are drawn:

Solid — The full path of the line is drawn.

DoubleDash — The full path of the line is drawn, but the even dashes are filled differently than the odd dashes (see *fill-style*), with *Butt cap-style* used where even and odd dashes meet.

OnOffDash — Only the even dashes are drawn, and *cap-style* applies to all internal ends of the individual dashes (except *NotLast* is treated as *Butt*).

cap-style defines how the endpoints of a path are drawn:

NotLast — The result is equivalent to *Butt*, except that for a *line-width* of 0 the final endpoint is not drawn.

Butt — The result is square at the endpoint (perpendicular to the slope of the line) with no projection beyond.

Round — The result is a circular arc with its diameter equal to *line-width*, centered on the endpoint; it is equivalent to *Butt* for *line-width* 0.

Projecting — The result is square at the end, but the path continues beyond the endpoint for a distance equal to half *line-width*; it is equivalent to *Butt* for *line-width* 0.

---

4. **Application Usage:** This mathematical model describes the pixels drawn for a wide line and does not imply that trigonometry is required to implement such a model. Real or fixed point arithmetic is recommended for computing the corners of the line endpoints for lines greater than one pixel in width.

join-style defines how corners are drawn for wide lines:

Miter	The outer edges of the two lines extend to meet at an angle. However, if the angle is less than 11 degrees, a *Bevel join-style* is used instead.
Round	The result is a circular arc with a diameter equal to *line-width*, centered on the joinpoint.
Bevel	The result is *Butt* endpoint styles, and then the triangular notch is filled.

For a line with coincident endpoints (x1=x2, y1=y2), when *cap-style* is applied to both endpoints, the semantics depend on *line-width* and *cap-style*:

NotLast	*thin*	Implementation-dependent. However, the intent is that nothing should be drawn.
Butt	*thin*	Implementation-dependent. However, the intent is that a single pixel should be drawn.
Round	*thin*	This is the same as *Butt* / *thin*.
Projecting	*thin*	This is the same as *Butt* / *thin*.
Butt	*wide*	Nothing is drawn.
Round	*wide*	The closed path is a circle, centered at the endpoint and with a diameter equal to *line-width*.
Projecting	*wide*	The closed path is a square, aligned with the coordinate axes, centered at the endpoint and with sides equal to *line-width*.

For a line with coincident endpoints (*x1=x2* and *y1=y2*), when *join-style* is applied at one or both endpoints, the effect is as if the line was removed from the overall path. However, if the total path consists of (or is reduced to) a single point joined with itself, the effect is the same as when *cap-style* is applied at both endpoints.

tile and *stipple* represent an infinite 2D plane, with the specified pattern replicated in all dimensions. When that plane is superimposed on the drawable for use in a graphics operation, the upper-left corner of some instance of the pattern is at the coordinates within the drawable specified by the tile/stipple origin. The *tile*, *stipple*, and clip origins are interpreted relative to the origin of whatever destination drawable is specified in a graphics request.

The *tile* pixmap must have the same root and depth as the graphics context being created, or a [MATCH] error results. The *stipple* pixmap must have depth one and must have the same root as the graphics context being created, or a [MATCH] error results. For *fill-style Stippled* (but not *OpaqueStippled*), *stipple* is tiled in a single plane and acts as an additional clip mask to be ANDed with *clip-mask*.

fill-style defines the contents of the source for line, text, and fill requests. For all text and fill requests (such as *PolyText8*, *PolyText16*, *PolyFillRectangle*, *FillPoly*, and *PolyFillArc*) as well as for line requests with *line-style Solid*, (for example, *PolyLine*, *PolySegment*, *PolyRectangle*, and *PolyArc*) and for the even dashes for line requests with *line-style* of *OnOffDash* or *DoubleDash*:

Solid	Foreground.
Tiled	Tile.
OpaqueStippled	A tile with the same width and height as *stipple* but with *background* everywhere *stipple* has a 0, and with *foreground* everywhere *stipple* has a 1.
Stippled	Foreground masked by *stipple*.

For the odd dashes for line requests with *line-style* of *DoubleDash*:

Solid	Background.
Tiled	Same as for even dashes.
OpaqueStippled	Same as for even dashes.
Stippled	Background masked by stipple.

The *dashes* value allowed here is a special case of the values settable by *SetDashes*. A *dashes* value of *N* is equivalent to specifying the two element list [*N*, *N*] in *SetDashes*. The value must be nonzero, or a [VALUE] error results. The meaning of *dash-offset* and *dashes* is explained in the *SetDashes* request.

clip-mask restricts writes to the destination drawable. Only pixels where *clip-mask* has bits set to 1 are drawn. Pixels are not drawn outside the area covered by *clip-mask* or where *clip-mask* has bits set to 0. *clip-mask* affects all graphics requests, but it does not clip sources. The *clip-mask* origin is interpreted relative to the origin of whatever destination drawable is specified in a graphics request. If a pixmap is specified as *clip-mask*, it must have depth 1 and have the same root as the graphics context being created, or a [MATCH] error results. If *clip-mask* is *None*, then pixels are always drawn, regardless of the clip origin. The *clip-mask* can also be set with the *SetClipRectangles* request.

For *ClipByChildren*, both source and destination windows are additionally clipped by all viewable *InputOutput* children. For *IncludeInferiors*, neither source nor destination window is clipped by inferiors. This results in including subwindow contents in the source and drawing through subwindow boundaries of the destination. The semantics of *IncludeInferiors*, when a mapped inferior has a different depth from the source or destination window, are unspecified.

fill-rule defines which pixels are inside (that is, are drawn) for paths given in *FillPoly* requests. *EvenOdd* means a point is inside if an infinite ray with the point as origin crosses the path an odd number of times. For *Winding*, a point is inside if an infinite ray with the point as origin crosses an unequal number of clockwise and counterclockwise directed path segments. A clockwise directed path segment is one that crosses the ray from left to right as observed from the point. A counter-clockwise segment is one that crosses the ray from right to left as observed from the point.

For both fill rules, a point is infinitely small and the path is an infinitely thin line. A pixel is inside if the center point of the pixel is inside and the center point is not on the boundary. If the center point is on the boundary, the pixel is inside if and only if the polygon interior is immediately to its right (x increasing direction). Pixels with centers along a horizontal edge are a special case and are inside if and only if the polygon interior is immediately below (y increasing direction).

arc-mode controls filling in the *PolyFillArc* request.

The *graphics-exposures* flag controls *GraphicsExpose* event generation for *CopyArea* and *CopyPlane* requests (and any similar requests defined by extensions).

The default component values are:

Component	Default
function	*Copy*
plane-mask	all ones

Component	Default
foreground	0
background	1
line-width	0
line-style	*Solid*
cap-style	*Butt*
join-style	*Miter*
fill-style	*Solid*
fill-rule	*EvenOdd*
arc-mode	*PieSlice*
tile	Pixmap of unspecified size filled with foreground pixel (that is, client specified pixel if any, else 0) (subsequent changes to foreground do not affect this pixmap).
stipple	Pixmap of unspecified size filled with ones.
tile-stipple-x-origin	0
tile-stipple-y-origin	0
font	An implementation-dependent font.
subwindow-mode	*ClipByChildren*
graphics-exposures	*True*
clip-x-origin	0
clip-y-origin	0
clip-mask	*None*
dash-offset	0
dashes	4 (that is, the list [4, 4])

It is unspecified whether storing a pixmap in a graphics context results in a copy being made. If the pixmap is later used as the destination for a graphics request, the change may be reflected in the graphics context. If the pixmap is used simultaneously in a graphics request as both a destination and as a tile or stipple, the results are unspecified.

CreateGlyphCursor

cid: CURSOR
source-font: FONT
mask-font: FONT or *None*
source-char, *mask-char*: CARD16
fore-red, *fore-green*, *fore-blue*: CARD16
back-red, *back-green*, *back-blue*: CARD16

Errors: [ALLOC], [FONT], [IDCHOICE], [VALUE]

This request is similar to *CreateCursor*, except the source and mask bitmaps are obtained from the specified font glyphs. *source-char* must be a defined glyph in *source-font*, and if *mask-font* is given, *mask-char* must be a defined glyph in *mask-font*, or a [VALUE] error results. The mask font and character are optional. The origins of the source and mask (if it is defined) glyphs are positioned coincidently and define the hotspot. The source and mask need not have the same bounding box metrics, and there is no restriction on the placement of the hotspot relative to the bounding boxes. If no mask is given, all pixels of the source are displayed. For 2-byte matrix fonts, the 16-bit value should be formed with *byte1* in the most-significant byte and *byte2* in the least-significant byte.

The components of the cursor may be transformed arbitrarily to meet display limitations.

The fonts can be freed immediately if no further explicit references to them are to be made.

CreatePixmap

pid: PIXMAP
drawable: DRAWABLE
depth: CARD8
width, *height*: CARD16

Errors: [ALLOC], [DRAWABLE], [IDCHOICE], [VALUE]

This request creates a pixmap and assigns the identifier *pid* to it. *width* and *height* must be nonzero, or a [VALUE] error results. *depth* must be one of the depths supported by the screen of *drawable*, or a [VALUE] error results. The initial contents of the pixmap are unspecified.

It is valid to pass an *InputOnly* window as a drawable to this request.

CreateWindow

wid, *parent*: WINDOW
class: {*InputOutput*, *InputOnly*, *CopyFromParent*}
depth: CARD8
visual: VISUALID or *CopyFromParent*
x, *y*: INT16
width, *height*, *border-width*: CARD16
value-mask: BITMASK
value-list: LISTofVALUE

Errors: [ALLOC], [COLORMAP], [CURSOR], [IDCHOICE], [MATCH], [PIXMAP], [VALUE], [WINDOW]

This request creates an unmapped window and assigns the identifier *wid* to it.

A *class* of *CopyFromParent* means the class is taken from the parent. A *depth* of 0 for class *InputOutput* or *CopyFromParent* means the depth is taken from the parent. A *visual* of *CopyFromParent* means the visual type is taken from the parent. For class *InputOutput*, *visual* and *depth* must be a combination supported for the screen, or a [MATCH] error results. *depth* need not be the same as the parent, but the parent must not be of class *InputOnly*, or a [MATCH] error results. For class *InputOnly*, *depth* must be 0, or a [MATCH] error results; and *visual* must be supported for the screen, or a [MATCH] error results. However, the parent can have any depth and class.

The server essentially acts as if *InputOnly* windows do not exist for the purposes of graphics requests, exposure processing, and *VisibilityNotify* events. An *InputOnly* window cannot be used as a drawable.

The coordinate system has the x axis horizontal and the y axis vertical, with the origin [0, 0] at the upper left. Coordinates are integral, in terms of pixels, and coincide with pixel centers. Each window and pixmap has its own coordinate system. For a window, the origin is inside the border at the inside upper left.

The *x* and *y* coordinates for the window are relative to the parent's origin and specify the position of the upper-left outer corner of the window (not the origin). *width* and *height* specify the inside size (not including the border) and must be nonzero, or a [VALUE] error results. *border-width* for an *InputOnly* window must be 0, or a [MATCH] error results.

The window is placed on top in the stacking order with respect to siblings.

value-mask and *value-list* specify attributes of the window that are to be explicitly initialized. The possible values, and default values when attributes are not explicitly initialized, are as follows:

Attribute	Type	Default
background-pixmap	PIXMAP or *None* or *ParentRelative*	*None*
background-pixel	CARD32	
border-pixmap	PIXMAP or *CopyFromParent*	*CopyFromParent*
border-pixel	CARD32	
bit-gravity	BITGRAVITY	*Forget*
win-gravity	WINGRAVITY	*NorthWest*
backing-store	{*NotUseful*, *WhenMapped*, *Always*}	*NotUseful*
backing-planes	CARD32	All ones
backing-pixel	CARD32	0
save-under	BOOL	*False*
event-mask	SETofEVENT	Empty set
do-not-propagate-mask	SETofDEVICEEVENT	Empty set
override-redirect	BOOL	*False*
colormap	COLORMAP or *CopyFromParent*	*CopyFromParent*
cursor	CURSOR or *None*	*None*

Only the following attributes are defined for *InputOnly* windows:

- *win-gravity*
- *event-mask*
- *do-not-propagate-mask*
- *override-redirect*
- *cursor*

It is a [MATCH] error to specify any other attributes for *InputOnly* windows.

If *background-pixmap* is given, it overrides the default *background-pixmap*. The background pixmap and the window must have the same root and the same depth, or a [MATCH] error results. If background *None* is specified, the window has no defined background. If background *ParentRelative* is specified, the parent's background is used, but the window must have the same depth as the parent, or a [MATCH] error results. If the parent has background *None*, then the window also has background *None*. A copy of the parent's background is not made. The parent's background is reexamined each time the window background is required.

If *background-pixel* is given, it overrides the default *background-pixmap* and any *background-pixmap* given explicitly. The *background-pixel* value is not range-checked but is truncated to the appropriate number of bits. For a *ParentRelative* background, the background tile origin always aligns with the parent's background tile origin. Otherwise, the background tile origin is always the window origin.

When no valid contents are available for regions of a window and the regions are either visible or the server is maintaining backing store, the server automatically tiles the regions with the window's background unless the window has a background of *None*. If the background is *None*, the screen contents of other windows of the same depth as the window are unchanged if the contents come from the parent of the window or an inferior of the parent; otherwise, the initial contents of the exposed regions are unspecified. *Expose* events are then generated for the regions, even if the background is *None*.

The border tile origin is always the same as the background tile origin. If *border-pixmap* is given, it overrides the default *border-pixmap*. The border pixmap and the window must have the same

root and the same depth, or a [MATCH] error results. If *CopyFromParent* is given, the parent's border pixmap is copied and subsequent changes to the parent's border attribute do not affect the child. In this case, the window must have the same depth as the parent, or a [MATCH] error results. The pixmap might be copied by sharing the same pixmap object between the child and parent or by making a complete copy of the pixmap contents. If *border-pixel* is given, it overrides the default *border-pixmap* and any *border-pixmap* given explicitly. Range checking is not performed on the *border-pixel* value; it is truncated to the appropriate number of bits.

Output to a window is always clipped to the inside of the window, so that the border is never affected.

bit-gravity defines which region of the window should be retained if the window is resized, and *win-gravity* defines how the window should be repositioned if the parent is resized (see *ConfigureWindow* request).

A *backing-store* of *WhenMapped* advises the server that maintaining contents of obscured regions when the window is mapped may be beneficial. A *backing-store* of *Always* advises the server that maintaining contents even when the window is unmapped may be beneficial. In this case, the server may generate an exposure event when the window is created. A value of *NotUseful* advises the server that maintaining contents is unnecessary, although a server may still maintain contents while the window is mapped.[5]

If *save-under* is *True*, the server is advised that when this window is mapped, saving the contents of windows it obscures may be beneficial.

When the contents of obscured regions of a window are being maintained, regions obscured by non-inferior windows are included in the destination (and source, when the window is the source) of graphics requests, but regions obscured by inferior windows are not included.

backing-planes indicates (with bits set to 1) which bit planes of the window hold dynamic data that must be preserved in backing store and during save-unders. *backing-pixel* specifies what value to use in planes not covered by *backing-planes*. The server is free to save only the specified bit planes in *backing-store* or *save-under* and regenerate the remaining planes with the specified pixel value. Any bits beyond the specified depth of the window in these values are ignored.

event-mask defines which events the client is interested in for this window (or for some event types, inferiors of the window). The *do-not-propagate-mask* defines which events should not be propagated to ancestor windows when no client has the event type selected in this window.

override-redirect specifies whether map and configure requests on this window should override a *SubstructureRedirect* on the parent, typically to inform a window manager not to tamper with the window.

colormap specifies the colormap that best reflects the true colors of the window. Servers capable of supporting multiple hardware colormaps may use this information, and window managers may use it for *InstallColormap* requests. *colormap* must have the same visual type and root as the window, or a [MATCH] error results. If *CopyFromParent* is specified, the parent's *colormap* is copied (subsequent changes to the parent's *colormap* attribute do not affect the child). However, the window must have the same visual type as the parent, or a [MATCH] error results; and the parent must not have a colormap of *None*, or a [MATCH] error results. For an explanation of

5. **Application Usage:** If the server maintains contents, then it should maintain complete contents, not just the region within the parent boundaries, even if the window is larger than its parent. While the server maintains contents, exposure events are not normally generated, but the server may stop maintaining contents at any time.

None, see *FreeColormap*. *colormap* is copied by sharing the colormap object between the child and the parent, not by making a complete copy of the colormap contents.

If a cursor is specified, it is used whenever the pointer is in the window. If *None* is specified, the parent's cursor is used when the pointer is in the window, and any change in the parent's cursor causes an immediate change in the displayed cursor.

This request generates a *CreateNotify* event.

The background and border pixmaps and the cursor may be freed immediately if no further explicit references to them are to be made.

The effect on the window state of subsequent drawing into the background or border pixmap is unspecified. It is unspecified whether the server makes a copy of the pixmap.

DeleteProperty

window: WINDOW
property: ATOM

Errors: [ATOM], [WINDOW]

If *property* exists, this request deletes it from *window* and generates a *PropertyNotify* event on *window*.

DestroySubwindows

window: WINDOW

Errors: [WINDOW]

This request performs a *DestroyWindow* request on all children of *window*, in bottom-to-top stacking order.

DestroyWindow

window: WINDOW

Errors: [WINDOW]

If *window* is mapped, an *UnmapWindow* request is performed automatically. *window* and all inferiors are then destroyed, and a *DestroyNotify* event is generated for each window. The ordering of the *DestroyNotify* events is such that for any given window, *DestroyNotify* is generated on all inferiors of the window before being generated on the window itself. The ordering among siblings and across subhierarchies is otherwise unspecified.

Normal exposure processing on formerly obscured windows is performed.

If the window is a root window, this request has no effect.

FillPoly

drawable: DRAWABLE
gc: GCONTEXT
shape: {*Complex*, *Nonconvex*, *Convex*}
coordinate-mode: {*Origin*, *Previous*}
points: LISTofPOINT

Errors: [DRAWABLE], [GCONTEXT], [MATCH], [VALUE]

This request fills the region closed by the specified path. The path is closed automatically if the last point in the list does not coincide with the first point. No pixel of the region is drawn more than once.

The first point is always relative to the drawable's origin. The rest are relative either to that origin or the previous point, depending on *coordinate-mode*

If *shape* is *Complex*, the path may self-intersect. Contiguous coincident points in the path are not treated as self-intersection. *Nonconvex* means the path does not self-intersect, but the shape is not wholly convex. If known by the client, specifying *Nonconvex* over *Complex* may improve performance. If *Nonconvex* is specified for a self-intersecting path, the graphics results are unspecified. *Convex* means that for every pair of points inside the polygon, the line segment connecting them does not intersect the path. If known by the client, specifying *Convex* may improve performance. If *Convex* is specified for a path that is not convex, the results are unspecified.

The following components of *gc* must be provided: *function*, *plane-mask*, *fill-style*, *fill-rule*, *subwindow-mode*, *clip-x-origin*, *clip-y-origin*, *clip-mask*.

The following mode-dependent components of *gc* must be provided: *foreground*, *background*, *tile*, *stipple*, *tile-stipple-x-origin*, *tile-stipple-y-origin*.

ForceScreenSaver

mode: {*Activate*, *Reset*}

Errors: [VALUE]

If *mode* is *Activate* and the screen-saver is currently deactivated, then the screen-saver is activated (even if the screen-saver has been disabled with a timeout value of 0). If *mode* is *Reset* and the screen-saver is currently enabled, then the screen-saver is deactivated (if it was activated), and the activation timer is reset to its initial state as if device input had just been received.

FreeColormap

cmap: COLORMAP

Errors: [COLORMAP]

This request deletes the association between the resource ID and *cmap* and frees the colormap storage. If *cmap* is an installed map for a screen, it is uninstalled. If *cmap* is defined as the colormap for a window (by means of *CreateWindow* or *ChangeWindowAttributes*), the colormap for the window is changed to *None*, and a *ColormapNotify* event is generated. The colors displayed for a window with a colormap of *None* are unspecified.

This request has no effect on a default colormap for a screen.

FreeColors

cmap: COLORMAP
pixels: LISTofCARD32
plane-mask: CARD32

Errors: [ACCESS], [COLORMAP], [VALUE]

plane-mask should not have any bits in common with any of the pixels. The set of all pixels is produced by ORing together subsets of *plane-mask* with the pixels. The request frees all of these pixels that were allocated by the client (using *AllocColor*, *AllocNamedColor*, *AllocColorCells*, and *AllocColorPlanes*).[6]

All specified pixels that are allocated by the client in *cmap* are freed, even if one or more pixels produce an error. A [VALUE] error is generated if a specified pixel is not a valid index into *cmap*. An [ACCESS] error is generated if a specified pixel is not allocated by the client (that is, is unallocated or is only allocated by another client) or if the colormap was created with all entries writable (using an *alloc* value of *All* in *CreateColormap*). If more than one pixel is in error, it is unspecified which pixel is reported.

FreeCursor

cursor: CURSOR

Errors: [CURSOR]

This request deletes the association between the resource ID and *cursor*. The *cursor* storage is freed when no other resource references it.

FreeGC

gc: GCONTEXT

Errors: [GCONTEXT]

This request deletes the association between the resource ID and *gc* and destroys the graphics context.

FreePixmap

pixmap: PIXMAP

Errors: [PIXMAP]

This request deletes the association between the resource ID and *pixmap*. The *pixmap* storage is freed when no other resource references it.

GetAtomName

atom: ATOM

→

name: STRING8

Errors: [ATOM]

This request returns the name for *atom*.

---

6. **Application Usage:** Freeing an individual pixel obtained from *AllocColorPlanes* may not actually let it be reused until all related pixels are freed. Similarly, a read-only entry is not actually freed until it has been freed by all clients, and if a client allocates the same read-only entry multiple times, it must free the entry that many times before the entry is actually freed.

GetFontPath

→
path: LISTofSTRING8

This request returns the current search path for fonts.

GetGeometry

drawable: DRAWABLE
→
root: WINDOW
depth: CARD8
x, *y*: INT16
width, *height*, *border-width*: CARD16

Errors: [DRAWABLE]

This request returns the root and current geometry of *drawable*. *depth* is the number of bits per pixel for the object. *x*, *y*, and *border-width* are always 0 for pixmaps. For a window, the *x* and *y* coordinates specify the upper-left outer corner of the window relative to its parent's origin, and *width* and *height* specify the inside size, not including the border.

It is valid to pass an *InputOnly* window as a drawable to this request.

GetImage

drawable: DRAWABLE
x, *y*: INT16
width, *height*: CARD16
plane-mask: CARD32
format: {*XYPixmap*, *ZPixmap*}
→
depth: CARD8
visual: VISUALID or *None*
data: LISTofBYTE

Errors: [DRAWABLE], [MATCH], [VALUE]

This request returns the contents of the given rectangle of *drawable* in *format*. The *x* and *y* coordinates are relative to the origin of *drawable* and define the upper-left corner of the rectangle. If *XYPixmap* is specified, only the bit planes specified in *plane-mask* are transmitted, with the planes appearing from most-significant to least-significant in bit order. If *ZPixmap* is specified, then bits in all planes not specified in *plane-mask* are transmitted as 0. Range checking is not performed on *plane-mask*; extraneous bits are ignored. The returned depth is as specified when *drawable* was created and is the same as a depth component in a FORMAT structure (in the connection setup), not a bits-per-pixel component. If *drawable* is a window, its visual type is returned. If *drawable* is a pixmap, the visual is *None*.

If *drawable* is a pixmap, then the given rectangle must be wholly contained within the pixmap, or a [MATCH] error results. If *drawable* is a window, the window must be viewable, and it must be the case that, if there were no overlapping or inferior windows, the specified rectangle of the window would be fully visible on the screen and wholly contained within the outside edges of the window, or a [MATCH] error results. The borders of the window can be included and read with this request. If the window has a backing store, then the *backing-store* contents are returned for regions of the window that are obscured by non-inferior windows; otherwise, the returned contents of such obscured regions are unspecified. The returned contents of visible regions of

inferiors of different depth other than the specified window are unspecified. The pointer cursor image is not included in the contents returned.

GetInputFocus

→
focus: WINDOW or *PointerRoot* or *None*
revert-to: {*Parent*, *PointerRoot*, *None*}

This request returns the current focus state.

GetKeyboardControl

→
key-click-percent: CARD8
bell-percent: CARD8
bell-pitch: CARD16
bell-duration: CARD16
led-mask: CARD32
global-auto-repeat: {*On*, *Off*}
auto-repeats: LISTofCARD8

This request returns the current control values for the keyboard. For the LEDs, the least-significant bit of *led-mask* corresponds to LED one, and each one bit in *led-mask* indicates an LED that is lit. *auto-repeats* is a bit vector; each one bit indicates that *auto-repeat* is enabled for the corresponding key. The vector is represented as 32 bytes. Byte N (from 0) contains the bits for keys 8N to 8N + 7, with the least-significant bit in the byte representing key 8N.

GetKeyboardMapping

first-keycode: KEYCODE
count: CARD8
→
keysyms-per-keycode: CARD8
keysyms: LISTofKEYSYM

Errors: [VALUE]

This request returns the symbols for the specified number of keycodes, starting with *first-keycode*. This must be greater than or equal to *min-keycode* as returned in the connection setup, or a [VALUE] error results; and:

first-keycode + *count* –1

must be less than or equal to *max-keycode* as returned in the connection setup, or a [VALUE] error results. The number of elements in the keysyms list is:

count * *keysyms-per-keycode*

and KEYSYM number N (counting from 0) for keycode K has an index (counting from 0) of:

(*K* – *first-keycode*) * *keysyms-per-keycode* + *N*

in keysyms. The server chooses a value of *keysyms-per-keycode* that is large enough to report all requested symbols. A special KEYSYM value of *NoSymbol* fills unused elements for individual keycodes.

GetModifierMapping

→
keycodes-per-modifier: CARD8
keycodes: LISTofKEYCODE

This request returns the keycodes of the keys being used as modifiers. The number of keycodes in the list is 8**keycodes-per-modifier*. The keycodes are divided into eight sets, with each set containing *keycodes-per-modifier* elements. The sets are assigned to the modifiers *Shift*, *Lock*, *Control*, *Mod1*, *Mod2*, *Mod3*, *Mod4*, and *Mod5*, in order. The server chooses a value of *keycodes-per-modifier* value and uses zeros to fill unused elements within each set. If only 0 values are given in a set, the use of the corresponding modifier has been disabled. The order of keycodes within each set is unspecified.

GetMotionEvents

start, *stop*: TIMESTAMP or *CurrentTime*
window: WINDOW
→
events: LISTofTIMECOORD

where:
TIMECOORD: [*x*, *y*: INT16
time: TIMESTAMP]

Errors: [WINDOW]

This request returns all events in the motion history buffer that occurred from *start* up to and including *stop*, and whose coordinates are within the borders of *window* at its present placement. The *x* and *y* coordinates are reported relative to the origin of *window*.

If *start* is later than *stop* or if *start* is in the future, no events are returned. If *stop* is in the future, it is equivalent to specifying *CurrentTime*.

GetPointerControl

→
acceleration-numerator, *acceleration-denominator*: CARD16
threshold: CARD16

This request returns the current acceleration and threshold for the pointer. The server may return different values of *acceleration-numerator* and *acceleration-denominator* from those most recently set with *ChangePointerControl*.

GetPointerMapping

→
map: LISTofCARD8

This request returns the current mapping of the pointer. Elements of the list are indexed starting from one. The length of the list indicates the number of physical buttons.

The nominal mapping for a pointer is the identity mapping: *map*[*i*]=*i*.

GetProperty

window: WINDOW
property: ATOM
type: ATOM or *AnyPropertyType*
long-offset, *long-length*: CARD32
delete: BOOL

→

type: ATOM or *None*
format: {0, 8, 16, 32}
bytes-after: CARD32
value: LISTofINT8 or LISTofINT16 or LISTofINT32

Errors: [ATOM], [VALUE], [WINDOW]

If *property* does not exist for *window*, then the return type is *None*, *format* and *bytes-after* are 0, and *value* is empty. The *delete* parameter is ignored in this case.

If *property* exists but its type does not match the specified type, then the return type is the actual type of the property, *format* is the actual format of the property (never 0), *bytes-after* is the length of the property in bytes (even if *format* is 16 or 32), and *value* is empty. The *delete* parameter is ignored in this case.

If *property* exists and either *AnyPropertyType* is specified or *type* matches the actual type of the property, then the return *type* is the actual type of the property, *format* is the actual format of the property (never 0), and *bytes-after* and *value* are as follows, given:

N Actual length of the stored property in bytes (even if *format* is 16 or 32).

I 4 * *long-offset*

T *N*–*I*

L MINIMUM(T, 4 * *long-length*)

A *N*–(*I*+*L*)

The returned *value* starts at byte index *I* in the property (indexing from 0), and its length in bytes is *L*. However, it is a [VALUE] error if *long-offset* is given such that *L* is negative. The value of *bytes-after* is *A*, giving the number of trailing unread bytes in the stored property. If *delete* is *True* and *bytes-after* is 0, the property is also deleted from the window, and a *PropertyNotify* event is generated on the window.

GetScreenSaver

→

timeout, *interval*: CARD16
prefer-blanking: {*Yes*, *No*}
allow-exposures: {*Yes*, *No*}

This request returns the current screen-saver control values.

GetSelectionOwner

selection: ATOM
→
owner: WINDOW or *None*

Errors: [ATOM]

This request returns any current owner window of *selection*, or *None* if there is no owner.

GetWindowAttributes

window: WINDOW
→
visual: VISUALID
class: {*InputOutput*, *InputOnly*}
bit-gravity: BITGRAVITY
win-gravity: WINGRAVITY
backing-store: {*NotUseful*, *WhenMapped*, *Always*}
backing-planes: CARD32
backing-pixel: CARD32
save-under: BOOL
colormap: COLORMAP or *None*
map-is-installed: BOOL
map-state: {*Unmapped*, *Unviewable*, *Viewable*}
all-event-masks, *your-event-mask*: SETofEVENT
do-not-propagate-mask: SETofDEVICEEVENT
override-redirect: BOOL

Errors: [WINDOW]

This request returns the current attributes of *window*. A window is *Unviewable* if it is mapped but some ancestor is unmapped. The returned *all-event-masks* is the inclusive-OR of all event masks selected on the window by clients, and *your-event-mask* is the event mask selected by the querying client.

GrabButton

modifiers: SETofKEYMASK or *AnyModifier*
button: BUTTON or *AnyButton*
grab-window: WINDOW
owner-events: BOOL
event-mask: SETofPOINTEREVENT
pointer-mode, *keyboard-mode*: {*Synchronous*, *Asynchronous*}
confine-to: WINDOW or *None*
cursor: CURSOR or *None*

Errors: [ACCESS], [CURSOR], [VALUE], [WINDOW]

This request establishes a passive grab. In the future, the pointer is actively grabbed as described in *GrabPointer*, the *last-pointer-grab* time is set to the time at which the button was pressed (as transmitted in the *ButtonPress* event), and the *ButtonPress* event is reported if all of the following conditions are true:

- The pointer is not grabbed and *button* is logically pressed when the specified modifier keys are logically down, and no other buttons or modifier keys are logically down.

- *grab-window* contains the pointer.
- Any *confine-to* window is viewable.
- A passive grab on the same button/key combination does not exist on any ancestor of *grab-window*.

The interpretation of the remaining parameters is the same as for *GrabPointer*. The active grab is terminated automatically when the logical state of the pointer has all buttons released, independent of the logical state of modifier keys. The logical state of a device (as seen by means of the protocol) may lag the physical state if device event processing is frozen.

This request overrides all previous passive grabs by the same client on the same button/key combinations on the same window. A modifier of *AnyModifier* is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). It is not required that all specified modifiers have currently assigned keycodes. A button of *AnyButton* is equivalent to issuing the request for all possible buttons. Otherwise, it is not required that the button specified currently be assigned to a physical button.

An [ACCESS] error is generated if some other client has already issued a *GrabButton* request with the same button/key combination on the same window. When using *AnyModifier* or *AnyButton*, the request fails completely (no grabs are established), and an [ACCESS] error is generated if there is a conflicting grab for any combination. The request has no effect on an active grab.

GrabKey

key: KEYCODE or *AnyKey*
modifiers: SETofKEYMASK or *AnyModifier*
grab-window: WINDOW
owner-events: BOOL
pointer-mode, *keyboard-mode*: {*Synchronous*, *Asynchronous*}

Errors: [ACCESS], [VALUE], [WINDOW]

This request establishes a passive grab on the keyboard. In the future, the keyboard is actively grabbed as described in *GrabKeyboard*, the *last-keyboard-grab* time is set to the time at which the key was pressed (as transmitted in the *KeyPress* event), and the *KeyPress* event is reported if all of the following conditions are true:

- The keyboard is not grabbed and *key* (which can itself be a modifier key) is logically pressed when *modifiers* are logically down, and no other modifier keys are logically down.
- Either *grab-window* is an ancestor of (or is) the focus window, or *grab-window* is a descendent of the focus window and contains the pointer.
- A passive grab on the same key combination does not exist on any ancestor of *grab-window*.

The interpretation of the remaining parameters is the same as for *GrabKeyboard*. The active grab is terminated automatically when the logical state of the keyboard has *key* released, independent of the logical state of modifier keys. The logical state of a device (as seen by means of the protocol) may lag the physical state if device event processing is frozen.

This request overrides all previous passive grabs by the same client on the same key combinations on the same window. A modifier of *AnyModifier* is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). It is not required that all modifiers specified have currently assigned keycodes. A key of *AnyKey* is equivalent to issuing the request for all possible keycodes. Otherwise, the key must be in the range specified by *min-keycode* and *max-keycode* in the connection setup, or a [VALUE] error

results.

An [ACCESS] error is generated if some other client has issued a *GrabKey* with the same key combination on the same window. When using *AnyModifier* or *AnyKey*, the request fails completely (no grabs are established), and an [ACCESS] error is generated if there is a conflicting grab for any combination.

GrabKeyboard

grab-window: WINDOW
owner-events: BOOL
pointer-mode, *keyboard-mode*: {*Synchronous*, *Asynchronous*}
time: TIMESTAMP or *CurrentTime*

→

status: {*Success*, *AlreadyGrabbed*, *Frozen*, *InvalidTime*, *NotViewable*}

Errors: [VALUE], [WINDOW]

This request actively grabs control of the keyboard. Further key events are reported only to the grabbing client. This request overrides any active keyboard grab by this client.

If *owner-events* is *False*, all generated key events are reported with respect to *grab-window*. If *owner-events* is *True* and if a generated key event would normally be reported to this client, it is reported normally. Otherwise, the event is reported with respect to *grab-window*. Both *KeyPress* and *KeyRelease* events are always reported, independent of any event selection made by the client.

If *keyboard-mode* is *Asynchronous*, keyboard event processing continues normally. If the keyboard is currently frozen by this client, then processing of keyboard events is resumed. If *keyboard-mode* is *Synchronous*, the state of the keyboard (as seen by means of the protocol) appears to freeze. No further keyboard events are generated by the server until the grabbing client issues a releasing *AllowEvents* request or until the keyboard grab is released. Actual keyboard changes are not lost while the keyboard is frozen. They are queued for later processing.

If *pointer-mode* is *Asynchronous*, pointer event processing is unaffected by activation of the grab. If *pointer-mode* is *Synchronous*, the state of the pointer (as seen by means of the protocol) appears to freeze. No further pointer events are generated by the server until the grabbing client issues a releasing *AllowEvents* request or until the keyboard grab is released. Actual pointer changes are not lost while the pointer is frozen. They are queued for later processing.

This request may generate *FocusIn* and *FocusOut* events.

The request fails with status *AlreadyGrabbed* if the keyboard is actively grabbed by some other client. The request fails with status *Frozen* if the keyboard is frozen by an active grab of another client. The request fails with status *NotViewable* if grab-window is unviewable. The request fails with status *InvalidTime* if *time* is earlier than the *last-keyboard-grab* time or later than the current server time. Otherwise, the *last-keyboard-grab* time is set to *time* with *CurrentTime* replaced by the current server time.

GrabPointer

grab-window: WINDOW
owner-events: BOOL
event-mask: SETofPOINTEREVENT
pointer-mode, *keyboard-mode*: {*Synchronous*, *Asynchronous*}
confine-to: WINDOW or *None*
cursor: CURSOR or *None*
time: TIMESTAMP or *CurrentTime*

→

status: {*Success*, *AlreadyGrabbed*, *Frozen*, *InvalidTime*, *NotViewable*}

Errors: [CURSOR], [VALUE], [WINDOW]

This request actively grabs control of the pointer. Further pointer events are only reported to the grabbing client. The request overrides any active pointer grab by this client.

If *owner-events* is *False*, all generated pointer events are reported with respect to *grab-window* and are only reported if selected by *event-mask*. If *owner-events* is *True* and a generated pointer event would normally be reported to this client, it is reported normally. Otherwise, the event is reported with respect to the *grab-window* and is only reported if selected by *event-mask*. For either value of *owner-events*, unreported events are discarded.

If *pointer-mode* is *Asynchronous*, pointer event processing continues normally. If the pointer is currently frozen by this client, then processing of pointer events is resumed. If *pointer-mode* is *Synchronous*, the state of the pointer (as seen by means of the protocol) appears to freeze, and no further pointer events are generated by the server until the grabbing client issues a releasing *AllowEvents* request or until the pointer grab is released. Actual pointer changes are not lost while the pointer is frozen. They are queued for later processing.

If *keyboard-mode* is *Asynchronous*, keyboard event processing is unaffected by activation of the grab. If *keyboard-mode* is *Synchronous*, the state of the keyboard (as seen by means of the protocol) appears to freeze, and no further keyboard events are generated by the server until the grabbing client issues a releasing *AllowEvents* request or until the pointer grab is released. Actual keyboard changes are not lost while the keyboard is frozen. They are queued for later processing.

If a cursor is specified, then it is displayed regardless of what window the pointer is in. If no cursor is specified, then when the pointer is in *grab-window* or one of its subwindows, the normal cursor for that window is displayed. Otherwise, the cursor for *grab-window* is displayed.

If *confine-to* is specified, then the pointer is restricted to stay contained in that window. The *confine-to* window need have no relationship to *grab-window*. If the pointer is not initially in *confine-to*, then it is warped automatically to the closest edge, generating enter/leave events normally, just before the grab activates. If *confine-to* is subsequently reconfigured, the pointer is warped automatically as necessary to keep it contained in the window.

This request generates *EnterNotify* and *LeaveNotify* events.

The request fails with status *AlreadyGrabbed* if the pointer is actively grabbed by some other client. The request fails with status *Frozen* if the pointer is frozen by an active grab of another client. The request fails with status *NotViewable* if *grab-window* or *confine-to* is unviewable or if *confine-to* lies completely outside the boundaries of the root window. The request fails with status *InvalidTime* if *time* is earlier than the *last-pointer-grab* time or later than the current server time. Otherwise, the *last-pointer-grab* time is set to the specified time, with *CurrentTime* replaced by the current server time.

GrabServer

(No request-specific parameters.)

This request disables processing of requests and close-downs on all connections other than the one this request arrived on.

ImageText8

drawable: DRAWABLE
gc: GCONTEXT
x, *y*: INT16
string: STRING8

Errors: [DRAWABLE], [GCONTEXT], [MATCH]

The *x* and *y* coordinates are relative to the drawable's origin and specify the baseline starting position (the initial character origin). The effect is first to fill a destination rectangle with the background pixel defined in *gc* and then to paint the text with the foreground pixel. The upper-left corner of the filled rectangle is at. [*x*, *y–font-ascent*], the width is *overall-width*, and the height is *font-ascent+font-descent*; where *overall-width*, *font-ascent*, and *font-descent* are the values *QueryTextExtents* would return for *gc* and *string*.

function and *fill-style* defined in *gc* are ignored for this request. The effective function is *Copy*, and the effective fill-style is *Solid*.

For fonts defined with 2-byte matrix indexing, each STRING8 byte is interpreted as a byte2 value of a CHAR2B with a byte1 value of 0.

The following components of *gc* must be provided: *plane-mask*, *foreground*, *background*, *font*, *subwindow-mode*, *clip-x-origin*, *clip-y-origin*, *clip-mask*.

ImageText16

drawable: DRAWABLE
gc: GCONTEXT
x, *y*: INT16
string: STRING16

Errors: [DRAWABLE], [GCONTEXT], [MATCH]

This request is similar to *ImageText8*, except 2-byte characters are used. For fonts defined with linear indexing rather than 2-byte matrix indexing, the server interprets each CHAR2B as a 16-bit number that has been transmitted most-significant byte first.

InstallColormap

cmap: COLORMAP

Errors: [COLORMAP]

This request makes *cmap* an installed map for its screen. All windows associated with *cmap* immediately display with correct colors. As a side effect, additional colormaps might be implicitly installed or uninstalled by the server. Which other colormaps get installed or uninstalled is implementation-dependent, except that the required list must remain installed.

If *cmap* is not already an installed map, a *ColormapNotify* event is generated on every window having *cmap* as an attribute. In addition, for every other colormap that is installed or uninstalled as a result of the request, a *ColormapNotify* event is generated on every window having that colormap as an attribute.

At any time, there is a subset of the installed maps that are conceptually an ordered list called the required list. The length of the required list is at most M, where M is the *min-installed-maps* specified for the screen in the connection setup. The required list is maintained as follows. When a colormap is an explicit parameter to *InstallColormap*, it is added to the head of the list; the list is truncated at the tail, if necessary, to keep its length to at most M. When a colormap is an explicit parameter to *UninstallColormap* and it is in the required list, it is removed from the list. A colormap is not added to the required list when it is installed implicitly by the server, and the server cannot implicitly uninstall a colormap that is in the required list.

Initially the default colormap for a screen is installed (but is not in the required list).

InternAtom

name: STRING8
only-if-exists: BOOL
→
atom: ATOM or *None*

Errors: [ALLOC], [VALUE]

This request returns the atom for *name*. If *only-if-exists* is *False*, then the atom is created if it does not exist.

The lifetime of an atom is not tied to the interning client. Atoms remain defined until server reset (see Section 2.6).

name is compared, case-sensitively, against stored atoms. If *name* does not use the ISO Latin-1 encoding, the results are implementation-dependent.

KillClient

resource: CARD32 or *AllTemporary*

Errors: [VALUE]

If *resource* is a valid resource ID, *KillClient* forces a close-down of the client that created *resource*. If the client has already terminated in either *RetainPermanent* or *RetainTemporary* mode, all the client's resources are destroyed (see Section 2.6). If *AllTemporary* is specified, then the resources of all clients that have terminated in *RetainTemporary* are destroyed.

ListExtensions

→
names: LISTofSTRING8

This request returns a list of all X Window System extensions supported by the server.

ListFonts

pattern: STRING8
max-names: CARD16
→
names: LISTofSTRING8

This request returns a list of up to *max-names* names of available fonts (as controlled by the font search path; see the *SetFontPath* request) that match *pattern*. In *pattern*, the character #0x3F (ASCII '?') matches any single character, and #0x2a (ASCII '*') matches any number of characters. The returned names are in lowercase.

pattern is compared, case-insensitively, to the names of available fonts. If *pattern* does not use the ISO Latin-1 encoding, the results are implementation-dependent.

ListFontsWithInfo

pattern: STRING8
max-names: CARD16
→+
name: STRING8
info: FONTINFO
replies-hint: CARD32

This request is similar to *ListFonts*, but it also returns information about each font. The information returned for each font is identical to what *QueryFont* would return except that the per-character metrics are not returned. This request can generate multiple replies. With each reply, *replies-hint* may indicate how many more fonts remain. This number is a hint only and may be larger or smaller than the number of fonts actually returned. A 0 value does not guarantee that the reply is the last one. After the font replies, a reply with a zero-length name is sent to indicate the end of the reply sequence.

ListHosts

→
mode: {*Enabled, Disabled*}
hosts: LISTofHOST

This request returns the hosts on the access control list and indicates whether use of the list at connection setup is currently enabled or disabled.

Each element of *hosts* is padded to a multiple of 4 bytes.

ListInstalledColormaps

window: WINDOW
→
cmaps: LISTofCOLORMAP

Errors: [WINDOW]

This request returns a list of the currently installed colormaps for the screen of *window*. The order of colormaps is not significant, and there is no explicit indication of the required list (see *InstallColormap* request).

ListProperties

window: WINDOW
→
atoms: LISTofATOM

Errors: [WINDOW]

This request returns the atoms of properties currently defined on *window*.

LookupColor

cmap: COLORMAP
name: STRING8
→
exact-red, *exact-green*, *exact-blue*: CARD16
visual-red, *visual-green*, *visual-blue*: CARD16

Errors: [COLORMAP], [NAME]

This request looks up the string name of a color with respect to the screen associated with *cmap* and returns both the exact color values and the closest values provided by the hardware with respect to the visual type of *cmap*.

name is compared, case-insensitively, to the valid color names. If *name* does not use the ISO Latin-1 encoding, the results are implementation-dependent.

MapWindow

window: WINDOW

Errors: [WINDOW]

If *window* is already mapped, this request has no effect.

If the *override-redirect* attribute of *window* is *False* and some other client has selected *SubstructureRedirect* on the parent, then a *MapRequest* event is generated, but *window* remains unmapped. Otherwise, *window* is mapped and a *MapNotify* event is generated.

If *window* is now viewable and its contents have been discarded, the window is tiled with its background (if no background is defined, the existing screen contents are not altered), and exposure events may be generated. If a backing store has been maintained while the window was unmapped, no exposure events are generated. If a backing store is now maintained, a full-window exposure is always generated. Otherwise, only visible regions may be reported. Similar tiling and exposure take place for any newly viewable inferiors.

MapSubwindows

window: WINDOW

Errors: [WINDOW]

This request performs a *MapWindow* request on all unmapped children of *window*, in top-to-bottom stacking order.

NoOperation

This request has no parameters and no results, but the *length* field can be nonzero, which lets the request be any multiple of 4 bytes in length. The server does not interpret the bytes of the request.[7]

7. **Application Usage:** This request can be used in its minimum 4-byte form as padding where necessary by client libraries that require requests to begin on 64-bit boundaries.

OpenFont

fid: FONT
name: STRING8

Errors: [ALLOC], [IDCHOICE], [NAME]

This request loads the font named *name*, if necessary, and associates identifier *fid* with it.

name is case-insensitive. *name* should use the ISO Latin-1 encoding and should not use the characters #0x3F (ASCII implementation-dependent.

Fonts are not associated with a particular screen and can be stored as a component of any graphics context.

PolyArc

drawable: DRAWABLE
gc: GCONTEXT
arcs: LISTofARC

Errors: [DRAWABLE], [GCONTEXT], [MATCH]

This request draws circular or elliptical arcs. Each arc is specified by a rectangle and two angles. The angles are signed integers in degrees scaled by 64, with positive indicating counterclockwise motion and negative indicating clockwise motion. The start of the arc is specified by *angle1* relative to the three-o'clock position from the center of the rectangle, and the path and extent of the arc is specified by *angle2* relative to the start of the arc. If the magnitude of *angle2* is greater than 360 degrees, it is truncated to 360 degrees. The *x* and *y* coordinates of the rectangle are relative to the origin of the drawable. For an arc specified as [*x*, *y*, *w*, *h*, *a1*, *a2*], the origin of the major and minor axes is at [*x*+(*w*/2), *y*+(*h*/2)], and the infinitely thin path describing the entire circle/ellipse intersects the horizontal axis at [*x*, *y*+(*h*/2)] and [*x*+*w*, *y*+(*h*/2)] and intersects the vertical axis at [*x*+(*w*/2), *y*] and [*x*+(*w*/2), *y*+*h*]. These coordinates are not necessarily integral; that is, they are not truncated to discrete coordinates.

For a wide line with line-width *lw*, the ideal bounding outlines for filling are given by the two infinitely thin paths consisting of all points whose perpendicular distance from a tangent to the path of the circle/ellipse is equal to *lw*/2 (which may be a fractional value). When the width and height of the arc are not equal and both are nonzero, then the actual bounding outlines are implementation-dependent. However, the computation of the shape and position of the bounding outlines (relative to the center of the arc) only depends on the width and height of the arc and the line-width.

cap-style is applied the same as for a line corresponding to the tangent of the circle/ellipse at the endpoint. When the angle of an arc face is not an integral multiple of 90 degrees, and the width and height of the arc are both are nonzero, then the shape and position of the cap at that face is implementation-dependent. However, for a *Butt* cap, the face is defined by a straight line, and the computation of the position (relative to the center of the arc) and the slope of the line only depends on the width and height of the arc and the angle of the arc face. For other cap styles, the computation of the position (relative to the center of the arc) and the shape of the cap only depends on the width and height of the arc, the line-width, the angle of the arc face, and the direction (clockwise or counter-clockwise) of the arc from the endpoint.

join-style is applied the same as for two lines corresponding to the tangents of the circles/ellipses at the join point. When the width and height of both arcs are nonzero, and the angle of either arc face is not an integral multiple of 90 degrees, then the shape of the join is implementation-dependent. However, the computation of the shape only depends on the width and height of each arc, *line-width*, the angles of the two arc faces, the direction (clockwise or counter-

clockwise) of the arcs from the join point, and the relative orientation of the two arc center points.

For an arc specified as [*x*, *y*, *w*, *h*, *a1*, *a2*], the angles must be specified in the effectively skewed coordinate system of the ellipse (for a circle, the angles and coordinate systems are identical). The relationship between these angles and angles expressed in the normal coordinate system of the screen is as follows:

skewed-angle = atan(tan(*normal-angle*) * *w* / *h*) + *adjust*

skewed-angle and *normal-angle* are expressed in radians (rather than in degrees scaled by 64) in the range [0, 2π). The atan returns a value in the range [–π / 2, π / 2]. *adjust* is:

0 For *normal-angle* in the range [0, π / 2).

π For *normal-angle* in the range [π / 2, (3π) / 2).

2π For *normal-angle* in the range [(3π) / 2, 2π).

The arcs are drawn in the order listed. If the last point in one arc coincides with the first point in the following arc, the two arcs join correctly. If the first point in the first arc coincides with the last point in the last arc, the two arcs join correctly. For any given arc, no pixel is drawn more than once. If two arcs join correctly and *line-width* is greater than 0 and the arcs intersect, no pixel is drawn more than once. Otherwise, the intersecting pixels of intersecting arcs are drawn multiple times. Specifying an arc with one endpoint and a clockwise extent draws the same pixels as specifying the other endpoint and an equivalent counterclockwise extent, except as it affects joins.

By specifying one axis to be 0, a horizontal or vertical line can be drawn.

Angles are computed based solely on the coordinate system, ignoring the aspect ratio.

The following components of *gc* must be provided: *function*, *plane-mask*, *line-width*, *line-style*, *cap-style*, *join-style*, *fill-style*, *subwindow-mode*, *clip-x-origin*, *clip-y-origin*, *clip-mask*.

The following mode-dependent components of *gc* must be provided: *foreground*, *background*, *tile*, *stipple*, *tile-stipple-x-origin*, *tile-stipple-y-origin*, *dash-offset*, *dashes*.

PolyFillArc

drawable: DRAWABLE
gc: GCONTEXT
arcs: LISTofARC

Errors: [DRAWABLE], [GCONTEXT], [MATCH]

For each arc, this request fills the region closed by the infinitely thin path described by the specified arc and one or two line segments, depending on *arc-mode*. For *arc-mode=Chord*, the single line segment joining the endpoints of the arc is used. For *arc-mode=PieSlice*, the two line segments joining the endpoints of the arc with the center point are used.

For an arc specified as [*x*, *y*, *w*, *h*, *a1*, *a2*], the origin of the major and minor axes is at [*x*+(*w* / 2), *y*+(*h* / 2)], and the infinitely thin path describing the entire circle / ellipse intersects the horizontal axis at [*x*, *y*+(*h* / 2)] and [*x*+*w*, *y*+(*h* / 2)] and intersects the vertical axis at [*x*+(*w* / 2), *y*] and [*x*+(*w* / 2), *y*+*h*]. These coordinates are not necessarily integral; that is, they are not truncated to discrete coordinates.

The arc angles are interpreted as specified in the *PolyArc* request. When the angle of an arc face is not an integral multiple of 90 degrees, then the precise endpoint on the arc is implementation-dependent. However, for *arc-mode=Chord*, the computation of the pair of endpoints (relative to

the center of the arc) only depends on the width and height of the arc and the angles of the two arc faces. For *arc-mode=PieSlice*, the computation of an endpoint only depends on the angle of the arc face for that endpoint and the ratio of the arc width to arc height.

The arcs are filled in the order listed. For any given arc, no pixel is drawn more than once. If regions intersect, the intersecting pixels are drawn multiple times.

The following components of *gc* must be provided: *function*, *plane-mask*, *fill-style*, *arc-mode*, *subwindow-mode*, *clip-x-origin*, *clip-y-origin*, *clip-mask*.

The following mode-dependent components of *gc* must be provided: *foreground*, *background*, *tile*, *stipple*, *tile-stipple-x-origin*, *tile-stipple-y-origin*.

PolyFillRectangle

drawable: DRAWABLE
gc: GCONTEXT
rectangles: LISTofRECTANGLE

Errors: [DRAWABLE], [GCONTEXT], [MATCH]

This request fills the specified rectangles, as if a 4-point *FillPoly* were specified for each rectangle:

[*x*, *y*]
[*x*+*width*, *y*]
[*x*+*width*, *y*+*height*]
[*x*, *y*+*height*]

The x and y coordinates of each rectangle are relative to the origin of *drawable* and define the upper-left corner of the rectangle.

The rectangles are drawn in the order listed. For any given rectangle, no pixel is drawn more than once. If rectangles intersect, the intersecting pixels are drawn multiple times.

The following components of *gc* must be provided: *function*, *plane-mask*, *fill-style*, *subwindow-mode*, *clip-x-origin*, *clip-y-origin*, *clip-mask*.

The following mode-dependent components of *gc* must be provided: *foreground*, *background*, *tile*, *stipple*, *tile-stipple-x-origin*, *tile-stipple-y-origin*.

PolyLine

drawable: DRAWABLE
gc: GCONTEXT
coordinate-mode: {*Origin*, *Previous*}
points: LISTofPOINT

Errors: [DRAWABLE], [GCONTEXT], [MATCH], [VALUE]

This request draws lines between each pair of points (*point*[*i*], *point*[*i*+1]). The lines are drawn in the order listed. The lines join correctly at all intermediate points, and if the first and last points coincide, the first and last lines also join correctly.

For any given line, no pixel is drawn more than once. If thin (0 line-width) lines intersect, the intersecting pixels are drawn multiple times. If wide lines intersect, the intersecting pixels are drawn only once, as though the entire *PolyLine* were a single filled shape.

The first point is always relative to the origin of *drawable*. The rest are relative either to that origin or the previous point, depending on *coordinate-mode*.

When either of the two lines involved in a *Bevel* join are neither vertical nor horizontal, then the slope and position of the line segment defining the bevel join edge is implementation-dependent. However, the computation of the slope and distance (relative to the join point) only depends on the line width and the slopes of the two lines.

The following components of *gc* must be provided: *function*, *plane-mask*, *line-width*, *line-style*, *cap-style*, *join-style*, *fill-style*, *subwindow-mode*, *clip-x-origin*, *clip-y-origin*, *clip-mask*.

The following mode-dependent components of *gc* must be provided: *foreground*, *background*, *tile*, *stipple*, *tile-stipple-x-origin*, *tile-stipple-y-origin*, *dash-offset*, *dashes*.

PolyPoint

drawable: DRAWABLE
gc: GCONTEXT
coordinate-mode: {*Origin*, *Previous*}
points: LISTofPOINT

Errors: [DRAWABLE], [GCONTEXT], [MATCH], [VALUE]

This request combines the foreground pixel in *gc* with the pixel at each point in *drawable*. The points are drawn in the order listed.

The first point is always relative to the origin of *drawable*. The rest are relative either to that origin or the previous point, depending on *coordinate-mode*.

The following components of *gc* must be provided: *function*, *plane-mask*, *foreground*, *subwindow-mode*, *clip-x-origin*, *clip-y-origin*, *clip-mask*.

PolyRectangle

drawable: DRAWABLE
gc: GCONTEXT
rectangles: LISTofRECTANGLE

Errors: [DRAWABLE], [GCONTEXT], [MATCH]

This request draws the outlines of the specified rectangles, as if a five-point *PolyLine* were specified for each rectangle:

[*x*, *y*] [*x+width*, *y*] [*x+width*, *y+height*] [*x*, *y+height*] [*x*, *y*]

The x and y coordinates of each rectangle are relative to the drawable's origin and define the upper-left corner of the rectangle.

The rectangles are drawn in the order listed. For any given rectangle, no pixel is drawn more than once. If rectangles intersect, the intersecting pixels are drawn multiple times.

The following components of *gc* must be provided: *function*, *plane-mask*, *line-width*, *line-style*, *cap-style*, *join-style*, *fill-style*, *subwindow-mode*, *clip-x-origin*, *clip-y-origin*, *clip-mask*.

The following mode-dependent components of *gc* must be provided: *foreground*, *background*, *tile*, *stipple*, *tile-stipple-x-origin*, *tile-stipple-y-origin*, *dash-offset*, *dashes*.

PolySegment

drawable: DRAWABLE
gc: GCONTEXT
segments: LISTofSEGMENT

where:
SEGMENT: [*x1*, *y1*, *x2*, *y2*: INT16]

Errors: [DRAWABLE], [GCONTEXT], [MATCH]

For each segment, this request draws a line between [*x1*, *y1*] and [*x2*, *y2*]. The lines are drawn in the order listed. No joining is performed at coincident endpoints. For any given line, no pixel is drawn more than once. If lines intersect, the intersecting pixels are drawn multiple times.

The following components of *gc* must be provided: *function*, *plane-mask*, *line-width*, *line-style*, *cap-style*, *fill-style*, *subwindow-mode*, *clip-x-origin*, *clip-y-origin*, *clip-mask*.

The following mode-dependent components of *gc* must be provided: *foreground*, *background*, *tile*, *stipple*, *tile-stipple-x-origin*, *tile-stipple-y-origin*, *dash-offset*, *dashes*.

PolyText8

drawable: DRAWABLE
gc: GCONTEXT
x, *y*: INT16
items: LISTofTEXTITEM8

Errors: [DRAWABLE], [FONT], [GCONTEXT], [MATCH]

The x and y coordinates are relative to the origin of *drawable* and specify the baseline starting position (the initial character origin). Each text item is processed in turn. A FONT item causes the font to be stored in *gc* and to be used for subsequent text. Switching among fonts does not affect the next character origin. A text element *delta* specifies an additional change in the position along the x axis before the string is drawn; the delta is always added to the character origin. Each character image, as defined by the font in *gc*, is treated as an additional mask for a fill operation on *drawable*.

All contained FONTs are always transmitted most-significant byte first.

If a [FONT] error is generated for an item, the previous items may have been drawn.

For fonts defined with 2-byte matrix indexing, each STRING8 byte is interpreted as a byte2 value of a CHAR2B with a byte1 value of 0.

The following components of *gc* must be provided: *function*, *plane-mask*, *fill-style*, *font*, *subwindow-mode*, *clip-x-origin*, *clip-y-origin*, *clip-mask*.

The following mode-dependent components of *gc* must be provided: *foreground*, *background*, *tile*, *stipple*, *tile-stipple-x-origin*, *tile-stipple-y-origin*.

PolyText16

drawable: DRAWABLE
gc: GCONTEXT
x, *y*: INT16
items: LISTofTEXTITEM16

Errors: [DRAWABLE], [FONT], [GCONTEXT], [MATCH]

This request is similar to *PolyText8*, except 2-byte characters are used. For fonts defined with linear indexing rather than 2-byte matrix indexing, the server interprets each CHAR2B as a 16-bit number that has been transmitted most-significant byte first.

PutImage

drawable: DRAWABLE
gc: GCONTEXT
depth: CARD8
width, *height*: CARD16
dst-x, *dst-y*: INT16
left-pad: CARD8
format: {*Bitmap, XYPixmap, ZPixmap* }
data: LISTofBYTE

Errors: [DRAWABLE], [GCONTEXT], [MATCH], [VALUE]

This request combines an image with a rectangle of *drawable*. The *dst-x* and *dst-y* coordinates are relative to the origin of *drawable*.

If *Bitmap* format is used, then *depth* must be one, or a [MATCH] error results; and the image must be in XY format. The foreground pixel in *gc* defines the source for bits set to 1 in the image, and the background pixel defines the source for the bits set to 0.

For *XYPixmap* and *ZPixmap*, *depth* must match the depth of *drawable*, or a [MATCH] error results. For *XYPixmap*, the image must be sent in XY format. For *ZPixmap*, the image must be sent in the Z format defined for *depth*.

left-pad must be 0 for *ZPixmap* format, or a [MATCH] error results. For *Bitmap* and *XYPixmap* format, *left-pad* must be less than *bitmap-scanline-pad* as given in the server connection setup information, or a [MATCH] error results. The server ignores the first *left-pad* bits in every scanline. The actual image begins that many bits into the data. The *width* parameter defines the width of the actual image and does not include *left-pad*.

The following components of *gc* must be provided: *function*, *plane-mask*, *subwindow-mode*, *clip-x-origin*, *clip-y-origin*, *clip-mask*.

The following mode-dependent components of *gc* must be provided: *foreground*, *background*.

QueryBestSize

class: {*Cursor, Tile, Stipple*}
drawable: DRAWABLE
width, *height*: CARD16
→
width, *height*: CARD16

Errors: [DRAWABLE], [MATCH], [VALUE]

This request returns the best size that is closest to the parameter size. For *Cursor*, this is the largest size that can be fully displayed. For *Tile*, this is the size that can be tiled fastest. For *Stipple*, this is the size that can be stippled fastest.

For *Cursor*, *drawable* indicates the desired screen. For *Tile* and *Stipple*, *drawable* indicates the screen and also possibly the window class and depth. An *InputOnly* window cannot be used as *drawable* for *Tile* or *Stipple*, or a [MATCH] error results.

QueryColors

cmap: COLORMAP
pixels: LISTofCARD32
→
colors: LISTofRGB

where:
RGB: [*red*, *green*, *blue*: CARD16]

Errors: [COLORMAP], [VALUE]

This request returns the implementation-dependent color values stored in *cmap* for *pixels*. The values returned for an unallocated entry are unspecified. A [VALUE] error is generated if a pixel is not a valid index into *cmap*. If more than one pixel is in error, it is unspecified which pixel is reported.

QueryExtension

name: STRING8
→
present: BOOL
major-opcode: CARD8
first-event: CARD8
first-error: CARD8

This request determines whether the X Window System extension named *name* is present. If so, the major opcode for the extension is returned, if it has one. Otherwise, 0 is returned. Any minor opcode and the request formats are specific to the extension. If the extension involves additional event types, the base event type code is returned. Otherwise, 0 is returned. The format of the events is specific to the extension. If the extension involves additional error codes, the base error code is returned. Otherwise, 0 is returned. The format of additional data in the errors is specific to the extension.

name is compared case-sensitively to the valid extension names. If *name* does not use the ISO Latin-1 encoding, the results are implementation-dependent.

QueryFont

font: FONTABLE
→
font-info: FONTINFO
char-infos: LISTofCHARINFO

where:

FONTINFO: [*draw-direction*: {*LeftToRight*, *RightToLeft*}
min-char-or-byte2, *max-char-or-byte2*: CARD16
min-byte1, *max-byte1*: CARD8
all-chars-exist: BOOL
default-char: CARD16
min-bounds: CHARINFO
max-bounds: CHARINFO
font-ascent: INT16
font-descent: INT16
properties: LISTofFONTPROP]

FONTPROP: [*name*: ATOM
value: <32-bit-value>]

CHARINFO: [*left-side-bearing*: INT16
right-side-bearing: INT16
character-width: INT16
ascent: INT16
descent: INT16
attributes: CARD16]

Errors: [FONT]

This request returns logical information about *font*. If a graphics context is given for *font*, the currently contained font is used.

draw-direction is a hint and indicates whether most *char-infos* have a positive, *LeftToRight*, or a negative, *RightToLeft*, character-width metric.

If *min-byte1* and *max-byte1* are both 0, then *min-char-or-byte2* specifies the linear character index corresponding to the first element of *char-infos*, and *max-char-or-byte2* specifies the linear character index of the last element. If either *min-byte1* or *max-byte1* are nonzero, then both *min-char-or-byte2* and *max-char-or-byte2* are less than 256, and the 2-byte character index values corresponding to *char-infos* element N (counting from 0) are:

byte1 = N / *d* + *min-byte1*
byte2 = N%*d* + *min-char-or-byte2*

where:

d = *max-char-or-byte2* – *min-char-or-byte2* + 1
/ = integer division
% = integer modulus

If *char-infos* has length 0, then *min-bounds* and *max-bounds* are identical, and the effective length of *char-infos* is:

L = *d* * (*max-byte1* – *min-byte1* + 1)

That is, all glyphs in the specified linear or matrix range have the same information, as given by *min-bounds* (and *max-bounds*). If *all-chars-exist* is *True*, then all glyphs in *char-infos* have nonzero

bounding boxes.

default-char specifies the character used for characters which are not defined or are nonexistent. This is a CARD16, not CHAR2B. For a font using 2-byte matrix format, *default-char* has byte1 in the most-significant byte and byte2 in the least-significant byte. If *default-char* itself specifies a character which is not defined or is nonexistent, then no printing is performed.

min-bounds and *max-bounds* contain the minimum and maximum values of each individual CHARINFO component over all *char-infos* (ignoring nonexistent characters). The bounding box of the font (that is, the smallest rectangle enclosing the shape obtained by superimposing all glyphs at the same origin [*x*, *y*]) has its upper-left coordinate at:

[*x+min-bounds.left-side-bearing*, *y–max-bounds.ascent*]

with a width of:

max-bounds.right-side-bearing–min-bounds.left-side-bearing

and a height of:

max-bounds.ascent+max-bounds.descent

font-ascent is the logical extent of the font above the baseline and is used for determining line spacing. Specific glyphs may extend beyond this. *font-descent* is the logical extent of the font at or below the baseline and is used for determining line spacing. Specific glyphs may extend beyond this. If the baseline is at y-coordinate *y*, then the logical extent of the font is inclusive between the y-coordinate values (*y–font-ascent*) and (*y+font-descent*–1).

A font is not guaranteed to have any properties. The interpretation of the property value (for example, INT32, CARD32) must be derived from *a priori* knowledge of the property. A basic set of font properties is specified in the **XLFD** specification (included in the **X Window System (X11R6): Fonts and Text** Technical Standard).

For a glyph origin at [*x*, *y*], the bounding box of a glyph (that is, the smallest rectangle enclosing the glyph's shape), described in terms of CHARINFO components, is a rectangle with its upper-left corner at:

[*x+left-side-bearing*, *y–ascent*]

with a width of:

right-side-bearing–left-side-bearing

and a height of:

ascent+descent

and the origin for the next glyph is defined to be:

[*x+character-width*, *y*]

The baseline is logically viewed as being just below non-descending glyphs (when descent is 0, only pixels with y-coordinates less than y are drawn) and that the origin is logically viewed as being coincident with the left edge of a non-kerned glyph (when left-side-bearing is 0, no pixels with x-coordinate less than x are drawn).

CHARINFO metric values can be negative.

A nonexistent character is represented with all CHARINFO components 0.

The interpretation of the per-character attributes field is implementation-dependent.

QueryKeymap

→
keys: LISTofCARD8

This request returns a bit vector for the logical state of the keyboard. Each bit set to 1 indicates that the corresponding key is currently pressed. The vector is represented as 32 bytes. Byte N (from 0) contains the bits for keys 8N to 8N + 7 with the least-significant bit in the byte representing key 8N. The logical state of a device (as seen by means of the protocol) may lag the physical state if device event processing is frozen.

QueryPointer

window: WINDOW
→
root: WINDOW
child: WINDOW or *None*
same-screen: BOOL
root-x, *root-y*, *win-x*, *win-y*: INT16
mask: SETofKEYBUTMASK

Errors: [WINDOW]

The root window the pointer is logically on and the pointer coordinates relative to the root's origin are returned. If *same-screen* is *False*, then the pointer is not on the same screen as the parameter window, *child* is *None*, and *win-x* and *win-y* are 0. If *same-screen* is *True*, then *win-x* and *win-y* are the pointer coordinates relative to the parameter window's origin, and *child* is the child containing the pointer, if any. The current logical state of the modifier keys and the buttons are also returned. The logical state of a device (as seen by means of the protocol) may lag the physical state if device event processing is frozen.

QueryTextExtents

font: FONTABLE
string: STRING16
→
draw-direction: {*LeftToRight*, *RightToLeft*}
font-ascent: INT16
font-descent: INT16
overall-ascent: INT16
overall-descent: INT16
overall-width: INT32
overall-left: INT32
overall-right: INT32

Errors: [FONT]

This request returns the logical extents of *string* when rendered in *font*. If a graphics context is given for *font*, the currently contained font is used. *draw-direction*, *font-ascent*, and *font-descent* are as described in *QueryFont*. *overall-ascent* is the maximum of the ascent metrics of all characters in the string, and *overall-descent* is the maximum of the descent metrics. *overall-width* is the sum of the character-width metrics of all characters in the string. For each character in the string, let W be the sum of the character-width metrics of all characters preceding it in the string, let L be the left-side-bearing metric of the character plus W, and let R be the right-side-bearing metric of the character plus W. *overall-left* is the minimum L of all characters in the string, and *overall-right* is the maximum R.

For fonts defined with linear indexing rather than 2-byte matrix indexing, the server interprets each CHAR2B as a 16-bit number that has been transmitted most-significant byte first.

Characters with all 0 metrics are ignored. If the font has no defined *default-char*, then characters in the string which are not defined are also ignored.

QueryTree

window: WINDOW
→
root: WINDOW
parent: WINDOW or *None*
children: LISTofWINDOW

Errors: [WINDOW]

This request returns the root, the parent, and the children of *window*. The children are listed in bottom-to-top stacking order.

RecolorCursor

cursor: CURSOR
fore-red, *fore-green*, *fore-blue*: CARD16
back-red, *back-green*, *back-blue*: CARD16

Errors: [CURSOR]

This request changes the color of *cursor*. If *cursor* is being displayed on a screen, the change is visible immediately.

ReparentWindow

window, *parent*: WINDOW
x, *y*: INT16

Errors: [MATCH], [WINDOW]

If *window* is mapped, an *UnmapWindow* request is performed automatically first. *window* is then removed from its current position in the hierarchy and is inserted as a child of *parent*. The *x* and *y* coordinates are relative to the origin of *parent* and specify the new position of the upper-left outer corner of *window*. *window* is placed on top in the stacking order with respect to siblings. A *ReparentNotify* event is then generated. The *override-redirect* attribute of *window* is passed on in this event; a value of *True* indicates that a window manager should not tamper with this window. Finally, if *window* was originally mapped, a *MapWindow* request is performed automatically.

Normal exposure processing on formerly obscured windows is performed. The server might not generate exposure events for regions from the initial unmap that are immediately obscured by the final map.

A [MATCH] error is generated if:

- *parent* is not on the same screen as the old parent.
- *parent* is *window* itself or an inferior of *window*.
- *parent* is *InputOnly* and *window* is not.
- *window* has a *ParentRelative* background, and *parent* is not the same depth as *window*.

RotateProperties

window: WINDOW
delta: INT16
properties: LISTofATOM

Errors: [ATOM], [MATCH], [WINDOW]

If the property names in the list are viewed as being numbered starting from 0, and there are N property names in the list, then the value associated with property name I becomes the value associated with property name (I + *delta*) mod N, for all I from 0 to N–1. The effect is to rotate the states by *delta* places around the virtual ring of property names (right for positive *delta*, left for negative *delta*).

If *delta* mod N is nonzero, a *PropertyNotify* event is generated for each property in the order listed.

If an atom occurs more than once in the list or no property with that name is defined for the window, a [MATCH] error is generated. If an [ATOM] or [MATCH] error is generated, no properties are changed.

SendEvent

destination: WINDOW or *PointerWindow* or *InputFocus*
propagate: BOOL
event-mask: SETofEVENT
event: 32-byte standard event format

Errors: [VALUE], [WINDOW]

If *PointerWindow* is specified, *destination* is replaced with the window that the pointer is in. If *InputFocus* is specified and the focus window contains the pointer, *destination* is replaced with the window that the pointer is in. Otherwise, *destination* is replaced with the focus window.

If *event-mask* is the empty set, then *event* is sent to the client that created the destination window. If that client no longer exists, no event is sent.

If *propagate* is *False*, then *event* is sent to every client selecting on *destination* any of the event types in *event-mask*.

If *propagate* is *True* and no clients have selected on *destination* any of the event types in *event-mask*, then *destination* is replaced with the closest ancestor of *destination* for which some client has selected a type in *event-mask* and no intervening window has that type in its *do-not-propagate-mask*. If no such window exists or if the window is an ancestor of the focus window and *InputFocus* was originally specified as *destination*, then *event* is not sent to any clients. Otherwise, *event* is reported to every client selecting on the final destination any of the types specified in *event-mask*.

The event code must be one of the core events or one of the events defined by an extension, or a [VALUE] error results, so that the server can correctly byte-swap the contents as necessary. The contents of the event are otherwise unaltered and unchecked by the server except to force on the most-significant bit of the event code and to set the sequence number in the event correctly.

Active grabs are ignored for this request.

SetAccessControl

mode: {*Enable*, *Disable*}

Errors: [ACCESS], [VALUE]

This request enables or disables the use of the access control list at connection setups.

The client must have been granted permission by a implementation-dependent method to execute this request, or an [ACCESS] error results.

SetClipRectangles

gc: GCONTEXT
clip-x-origin, *clip-y-origin*: INT16
rectangles: LISTofRECTANGLE
ordering: {*UnSorted*, *YSorted*, *YXSorted*, *YXBanded*}

Errors: [ALLOC], [GCONTEXT], [MATCH], [VALUE]

This request changes *clip-mask* in *gc* to the specified list of rectangles and sets the clip origin. Output is clipped to remain contained within the rectangles. The clip origin is interpreted relative to the origin of whatever destination drawable is specified in a graphics request. The rectangle coordinates are interpreted relative to the clip origin. The rectangles should be non-intersecting, or graphics results are unspecified. The list of rectangles can be empty, which effectively disables output. This is the opposite of passing *None* as *clip-mask* in *CreateGC* and *ChangeGC*.

If the client specifies ordering relations on the rectangles with *ordering*, it may provide faster operation by the server. If an incorrect ordering is specified, the server may generate a [MATCH] error; if no error is generated, the graphics results are unspecified. *UnSorted* means that the rectangles are in unspecified order. *YSorted* means that the rectangles are non-decreasing in their Y origin. *YXSorted* additionally constrains *YSorted* order in that all rectangles with an equal Y origin are non-decreasing in their X origin. *YXBanded* additionally constrains *YXSorted* by requiring that, for every possible Y scanline, all rectangles that include that scanline have identical Y origins and Y extents.

SetCloseDownMode

mode: {*Destroy*, *RetainPermanent*, *RetainTemporary*}

Errors: [VALUE]

This request defines what happens to the client's resources at connection close. A connection starts in *Destroy* mode.

SetDashes

gc: GCONTEXT
dash-offset: CARD16
dashes: LISTofCARD8

Errors: [ALLOC], [GCONTEXT], [VALUE]

This request sets *dash-offset* and *dashes* in *gc* for dashed line styles. *dashes* cannot be empty, or a [VALUE] error results. Specifying an odd-length list is equivalent to specifying the same list concatenated with itself to produce an even-length list. The initial and alternating elements of dashes are the even dashes; the others are the odd dashes. Each element specifies a dash length in pixels. All of the elements must be nonzero or a [VALUE] error results. *dash-offset* specifies

how many pixels into *dashes* the pattern begins in any single graphics request. Dashing is continuous through path elements combined with a *join-style* but is reset to *dash-offset* between each sequence of joined lines.

The unit of measure for dashes is the same for the ordinary coordinate system for horizontal and vertical lines; for other lines, it is unspecified.

For any graphics primitive, the computation of the endpoint of an individual dash only depends on the geometry of the primitive, the start position of the dash, the direction of the dash, and the dash length.

For any graphics primitive, the total set of pixels used to render the primitive (both even and odd numbered dash elements) with *line-style=DoubleDash* is the same as the set of pixels used to render the primitive with *line-style=Solid*.

For any graphics primitive, if the primitive is drawn with *OnOffDash* or *DoubleDash line-style* unclipped at position [*x, y*] and again at position [*x+dx, y+dy*], then a point [*x1, y1*] is included in a dash in the first instance if and only if the point [*x1+dx, y1+dy*] is included in the dash in the second instance. In addition, the effective set of points comprising a dash cannot be affected by clipping. A point is included in a clipped dash if and only if the point lies inside the clipping region and the point would be included in the dash when drawn unclipped.

SetFontPath

path: LISTofSTRING8

Errors: [VALUE]

This request defines the search path for font lookup. There is only one search path per server, not one per client. The interpretation of the strings is implementation-dependent, but the strings are intended to specify directories to be searched in the order listed.

Setting *path* to the empty list restores the default path defined for the server.

The server flushes all cached information about fonts that have no explicit resource IDs allocated.

The meaning of an error from this request is implementation-dependent.

SetInputFocus

focus: WINDOW or *PointerRoot* or *None*
revert-to: {*Parent*, *PointerRoot*, *None*}
time: TIMESTAMP or *CurrentTime*

Errors: [MATCH], [VALUE], [WINDOW]

This request changes the input focus and the *last-focus-change* time. The request has no effect if *time* is earlier than the current *last-focus-change* time or is later than the current server time. Otherwise, the *last-focus-change* time is set to *time* with *CurrentTime* replaced by the current server time.

If *None* is specified as *focus*, all keyboard events are discarded until a new focus window is set. In this case, *revert-to* is ignored.

If a window is specified as the focus, it becomes the keyboard's focus window. If a generated keyboard event would normally be reported to this window or one of its inferiors, the event is reported normally. Otherwise, the event is reported with respect to the focus window.

If *PointerRoot* is specified as the focus, the focus window is dynamically taken to be the root window of whatever screen the pointer is on at each keyboard event. In this case, *revert-to* is ignored.

This request generates *FocusIn* and *FocusOut* events.

focus must be viewable at the time of the request, or a [MATCH] error results. If the focus window later becomes unviewable, the new focus window depends on *revert-to*. If *revert-to* is *Parent*, the focus reverts to the parent (or the closest viewable ancestor) and the new *revert-to* value is taken to be *None*. If *revert-to* is *PointerRoot* or *None*, the focus reverts to that value. When the focus reverts, *FocusIn* and *FocusOut* events are generated, but the *last-focus-change* time is not affected.

SetModifierMapping

keycodes-per-modifier: CARD8
keycodes: LISTofKEYCODE

→

status: {*Success*, *Busy* , *Failed*}

Errors: [ALLOC], [VALUE]

This request specifies the keycodes (if any) of the keys to be used as modifiers. The number of keycodes in the list must be 8* *keycodes-per-modifier*, or a [LENGTH] error results. The keycodes are divided into eight sets, with each set containing *keycodes-per-modifier* elements. The sets are assigned to the modifiers *Shift*, *Lock*, *Control*, *Mod1*, *Mod2*, *Mod3*, *Mod4*, and *Mod5*, in order. Only nonzero keycode values are used within each set; 0 values are ignored. All of the nonzero keycodes must be in the range specified by *min-keycode* and *max-keycode* in the connection setup, or a [VALUE] error results. The order of keycodes within a set does not matter. If no nonzero values are specified in a set, the use of the corresponding modifier is disabled, and the modifier bit is always 0. Otherwise, the modifier bit is 1 whenever at least one of the keys in the corresponding set is in the down position.

There may be implementation-dependent restrictions on how modifiers can be changed (for example, if certain keys do not generate up transitions in hardware, if auto-repeat cannot be disabled on certain keys, or if multiple keys per modifier are not supported). The status reply is *Failed* if some such restriction is violated, and no modifier is changed.

If the new nonzero keycodes specified for a modifier differ from those currently defined and any (current or new) keys for that modifier are logically in the down state, then the status reply is *Busy*, and none of the modifiers is changed.

This request generates a *MappingNotify* event on a *Success* status.

SetPointerMapping

map: LISTofCARD8

→

status: {*Success*, *Busy*}

Errors: [VALUE]

This request sets the mapping of the pointer. Elements of the list are indexed starting from one. The length of the list must be the same as *GetPointerMapping* would return, or a [VALUE] error results. The index is a core button number, and the element of the list defines the effective number.

A 0 element disables a button. Elements are not restricted in value by the number of physical buttons, but no two elements can have the same nonzero value, or a [VALUE] error results.

If any of the buttons to be altered are logically in the down state, the status reply is *Busy*, and the mapping is not changed.

This request generates a *MappingNotify* event on a *Success* status.

SetScreenSaver

timeout, *interval*: INT16
prefer-blanking: {*Yes*, *No*, *Default*}
allow-exposures: {*Yes*, *No*, *Default*}

Errors: [VALUE]

timeout and *interval* are specified in seconds; setting a value to –1 restores the default. Other negative values generate a [VALUE] error. If *timeout* is 0, the screen-saver is disabled (but an activated screen-saver is not deactivated). If *timeout* is nonzero, the screen-saver is enabled. Once the screen-saver is enabled, if no input from the keyboard or pointer is generated for *timeout* seconds, the screen-saver is activated. For each screen, if blanking is preferred and the hardware supports video blanking, the screen goes blank. Otherwise, if either exposures are allowed or the screen can be regenerated without sending exposure events to clients, the screen is changed in an implementation-dependent fashion. Otherwise, the state of the screens does not change, and screen-saver is not activated. At the next keyboard or pointer input or at the next *ForceScreenSaver* with mode *Reset*, screen-saver is deactivated, and all screen states are restored.[8]

SetSelectionOwner

selection: ATOM
owner: WINDOW or *None*
time: TIMESTAMP or *CurrentTime*

Errors: [ATOM], [WINDOW]

This request changes the owner, owner window, and *last-change-time* of *selection*. This request has no effect if *time* is earlier than the current *last-change-time* of *selection* or is later than the current server time. Otherwise, the *last-change-time* is set to *time* with *CurrentTime* replaced by the current server time. If *owner* is *None*, then the owner of the selection becomes *None*. Otherwise, the owner of the selection becomes the client executing the request. If the new owner (whether a client or *None*) is not the same as the current owner and the current owner is not *None*, then the current owner is sent a *SelectionClear* event.

If the client that is the owner of a selection is later terminated, or if the owner window it has specified in the request is later destroyed, then the owner of the selection automatically reverts to *None*, but the *last-change-time* is not affected.

The *selection* atom is uninterpreted by the server. The owner window is returned by the *GetSelectionOwner* request and is reported in *SelectionRequest* and *SelectionClear* events.

8. **Application Usage:** If the implementation-dependent screen-saver method supports periodic change, *interval* serves as a hint on how long the change period should be. 0 hints that no periodic change should be made. Examples of periodic changes to the screen include scrambling the colormap, moving an icon image about the screen, or tiling the screen with the root window background tile with a changed origin.

Selections are global to the server.

StoreColors

cmap: COLORMAP
items: LISTofCOLORITEM

Errors: [ACCESS], [COLORMAP], [VALUE]

This request changes the colormap entries of the specified pixels. The *do-red*, *do-green*, and *do-blue* fields indicate which components should change. If *cmap* is an installed map for its screen, the changes are visible immediately.

All specified pixels that are allocated writable in *cmap* (by any client) are changed, even if one or more pixels produce an error. A [VALUE] error is generated if a specified pixel is not a valid index into *cmap*, and an [ACCESS] error is generated if a specified pixel is unallocated or is allocated read-only. If more than one pixel is in error, it is unspecified which pixel is reported.

StoreNamedColor

cmap: COLORMAP
pixel: CARD32
name: STRING8
do-red, *do-green*, *do-blue*: BOOL

Errors: [ACCESS], [COLORMAP], [NAME], [VALUE]

This request looks up the color named *name* with respect to the screen associated with *cmap* and then does a *StoreColors* in *cmap*. The [ACCESS] and [VALUE] errors are the same as in *StoreColors*.

If *name* does not use the ISO Latin-1 encoding, the results are implementation-dependent.

TranslateCoordinates

src-window, *dst-window*: WINDOW
src-x, *src-y*: INT16
→
same-screen: BOOL
child: WINDOW or *None*
dst-x, *dst-y*: INT16

Errors: [WINDOW]

The *src-x* and *src-y* coordinates are taken relative to the origin of *src-window* and are returned as *dst-x* and *dst-y* coordinates relative to the origin of *dst-window*. If *same-screen* is *False*, then *src-window* and *dst-window* are on different screens, and *dst-x* and *dst-y* are 0. If the coordinates are contained in a mapped child of *dst-window*, then that child is returned.

UngrabButton

modifiers: SETofKEYMASK or *AnyModifier*
button: BUTTON or *AnyButton*
grab-window: WINDOW

Errors: [VALUE], [WINDOW]

This request releases the passive button/key combination on *grab-window* if it was grabbed by this client. If *modifiers* is *AnyModifier*, it is equivalent to issuing the request for all possible

modifier combinations (including the combination of no modifiers). A button of *AnyButton* is equivalent to issuing the request for all possible buttons. The request has no effect on an active grab.

UngrabKey

key: KEYCODE or *AnyKey*
modifiers: SETofKEYMASK or *AnyModifier*
grab-window: WINDOW

Errors: [VALUE], [WINDOW]

This request releases the key combination on *grab-window* if it was grabbed by this client. If *modifiers* is *AnyModifier*, it is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). A key of *AnyKey* is equivalent to issuing the request for all possible keycodes. This request has no effect on an active grab.

UngrabKeyboard

time: TIMESTAMP or *CurrentTime*

This request releases the keyboard if this client has it actively grabbed (as a result of either *GrabKeyboard* or *GrabKey*) and releases any queued events. The request has no effect if *time* is earlier than the *last-keyboard-grab* time or is later than the current server time.

This request generates *FocusIn* and *FocusOut* events.

An *UngrabKeyboard* is performed automatically if the event window for an active keyboard grab becomes unviewable.

UngrabPointer

time: TIMESTAMP or *CurrentTime*

This request releases the pointer if this client has it actively grabbed (from either *GrabPointer* or *GrabButton* or from a normal button press) and releases any queued events. The request has no effect if *time* is earlier than the *last-pointer-grab* time or is later than the current server time.

This request generates *EnterNotify* and *LeaveNotify* events.

An *UngrabPointer* request is performed automatically if the event window or *confine-to* window for an active pointer grab becomes unviewable or if window reconfiguration causes the *confine-to* window to lie completely outside the boundaries of the root window.

UngrabServer

(No request-specific parameters.)

This request restarts processing of requests and close-downs on other connections.

UninstallColormap

cmap: COLORMAP

Errors: [COLORMAP]

If *cmap* is on the required list for its screen (see *InstallColormap* request), it is removed from the list. Also, *cmap* may be uninstalled, and additional colormaps may be implicitly installed or uninstalled. Which colormaps get installed or uninstalled is implementation-dependent, except that the required list must remain installed.

If *cmap* becomes uninstalled, a *ColormapNotify* event is generated on every window having *cmap* as an attribute. In addition, for every other colormap that is installed or uninstalled as a result of the request, a *ColormapNotify* event is generated on every window having that colormap as an attribute.

UnmapSubwindows

window: WINDOW

Errors: [WINDOW]

This request performs an *UnmapWindow* request on all mapped children of *window*, in bottom-to-top stacking order.

UnmapWindow

window: WINDOW

Errors: [WINDOW]

If the window is already unmapped, this request has no effect. Otherwise, the window is unmapped, and an *UnmapNotify* event is generated. Normal exposure processing on formerly obscured windows is performed.

WarpPointer

src-window: WINDOW or *None*
dst-window: WINDOW or *None*
src-x, src-y: INT16
src-width, src-height: CARD16
dst-x, dst-y: INT16

Errors: [WINDOW]

If *dst-window* is *None*, this request moves the pointer by offsets [*dst-x, dst-y*] relative to the current position of the pointer. If *dst-window* is a window, this request moves the pointer to [*dst-x, dst-y*] relative to the origin of *dst-window*. However, if *src-window* is not *None*, the move only takes place if *src-window* contains the pointer and the pointer is contained in the specified rectangle of *src-window*.

The *src-x* and *src-y* coordinates are relative to the origin of *src-window*. If *src-height* is 0, it is replaced with the current height of *src-window* minus *src-y*. If *src-width* is 0, it is replaced with the current width of *src-window* minus *src-x*.

This request cannot be used to move the pointer outside the *confine-to* window of an active pointer grab. An attempt only moves the pointer as far as the closest edge of the *confine-to* window.

This request generates events as if the user had instantaneously moved the pointer.

Chapter 4

Events

Section 4.1 lists alphabetically each type of core event that a server can send to a client, following the syntactic conventions set out in Section 1.3 on page 3.

Section 4.2 on page 93 specifies the effects when a button is pressed and no active grab is in progress.

4.1 Alphabetical List of Events

ButtonPress, ButtonRelease

root, *event*: WINDOW
child: WINDOW or *None*
same-screen: BOOL
root-x, *root-y*, *event-x*, *event-y*: INT16
detail: BUTTON
state: SETofKEYBUTMASK
time: TIMESTAMP

These events are generated when a button logically changes state. The generation of these logical changes may lag the physical changes if device event processing is frozen. The source of the event is the window the pointer is in. The *event* window is found by starting with the source window and looking up the hierarchy for the first window on which any client has selected interest in the event (provided no intervening window prohibits event generation by including the event type in its *do-not-propagate-mask*). The actual window used for reporting can be modified by active grabs and, in the case of keyboard events, can be modified by the focus window.

root is the root window of the source window, and *root-x* and *root-y* are the pointer coordinates relative to the root's origin at the time of the event.

If *event* is on the same screen as *root*, then *event-x* and *event-y* are the pointer coordinates relative to the event window's origin. Otherwise, *event-x* and *event-y* are 0.

If the source window is an inferior of *event*, then *child* is set to the child of *event* that is an ancestor of (or is) the source window. Otherwise, it is set to *None*. *state* gives the logical state of the buttons and modifier keys just before the event.

CirculateNotify

event, *window*: WINDOW
place: {*Top*, *Bottom*}

This event is reported to clients selecting *StructureNotify* on the window and to clients selecting *SubstructureNotify* on the parent. It is generated when the window is actually restacked from a *CirculateWindow* request. *event* is the window on which the event was generated, and *window* is the window that is restacked. If *place* is *Top*, the window is now on top of all siblings. Otherwise, it is below all siblings.

CirculateRequest

parent, *window*: WINDOW
place: {*Top*, *Bottom*}

This event is reported to the client selecting *SubstructureRedirect* on *parent* and is generated when a *CirculateWindow* request is issued on *parent* and a window actually needs to be restacked. *window* specifies the window to be restacked, and *place* specifies the new position in the stacking order.

ClientMessage

window: WINDOW
type: ATOM
format: {8, 16, 32}
data: LISTofINT8 or LISTofINT16 or LISTofINT32

This event is only generated by clients using *SendEvent*. *type* specifies how *data* is to be interpreted by the receiving client; the server does not interpret *type* or *data*. *format* specifies whether the data should be viewed as a list of 8-bit, 16-bit, or 32-bit quantities, so that the server can correctly byte-swap, as necessary. *data* consists of either twenty 8-bit values or ten 16-bit values or five 32-bit values, although particular message types might not make use of all of these values.

ColormapNotify

window: WINDOW
colormap: COLORMAP or *None*
new: BOOL
state: {*Installed*, *Uninstalled*}

This event is reported to clients selecting *ColormapChange* on the window. It is generated with *new=True* when the colormap attribute of *window* is changed, and with *new=False* when the colormap of *window* is installed or uninstalled. In either case, *state* indicates whether the colormap is currently installed.

ConfigureNotify

event, *window*: WINDOW
x, *y*: INT16
width, *height*, *border-width*: CARD16
above-sibling: WINDOW or *None*
override-redirect: BOOL

This event is reported to clients selecting *StructureNotify* on the window and to clients selecting *SubstructureNotify* on the parent. It is generated when a *ConfigureWindow* request actually changes the state of the window. *event* is the window on which the event was generated, and *window* is the window that is changed. The *x* and *y* coordinates are relative to the new parent's origin and specify the position of the upper-left outer corner of the window. *width* and *height* specify the inside size, not including the border. If *above-sibling* is *None*, then the window is on the bottom of the stack with respect to siblings. Otherwise, the window is immediately on top of the specified sibling. The *override-redirect* flag is from the window's attribute.

ConfigureRequest

parent, *window*: WINDOW
x, *y*: INT16
width, *height*, *border-width*: CARD16
sibling: WINDOW or *None*
stack-mode: {*Above*, *Below*, *TopIf*, *BottomIf*, *Opposite*}
value-mask: BITMASK

This event is reported to the client selecting *SubstructureRedirect* on the parent and is generated when a *ConfigureWindow* request is issued on the window by some other client. *value-mask* indicates the components specified in the request. *value-mask* and the corresponding values are reported as given in the request. The remaining values are filled in from the current geometry of the window, except in the case of *sibling* and *stack-mode*, which are reported as *None* and *Above*, respectively, if not given in the request.

CreateNotify

parent, *window*: WINDOW
x, *y*: INT16
width, *height*, *border-width*: CARD16
override-redirect: BOOL

This event is reported to clients selecting *SubstructureNotify* on the parent and is generated when the window is created. The parameters are as in the *CreateWindow* request.

DestroyNotify

event, *window*: WINDOW

This event is reported to clients selecting *StructureNotify* on the window and to clients selecting *SubstructureNotify* on the parent. It is generated when the window is destroyed. *event* is the window on which the event was generated, and *window* is the window that is destroyed.

The ordering of the *DestroyNotify* events is such that for any given window, *DestroyNotify* is generated on all inferiors of the window before being generated on the window itself. The ordering among siblings and across subhierarchies is otherwise unspecified.

EnterNotify, LeaveNotify

root, *event*: WINDOW
child: WINDOW or *None*
same-screen: BOOL
root-x, *root-y*, *event-x*, *event-y*: INT16
mode: {*Normal*, *Grab*, *Ungrab*}
detail: {*Ancestor*, *Virtual*, *Inferior*, *Nonlinear*, *NonlinearVirtual*}
focus: BOOL
state: SETofKEYBUTMASK
time: TIMESTAMP

If pointer motion or window hierarchy change causes the pointer to be in a different window than before, *EnterNotify* and *LeaveNotify* events are generated instead of a *MotionNotify* event. Only clients selecting *EnterWindow* on a window receive *EnterNotify* events, and only clients selecting *LeaveWindow* receive *LeaveNotify* events. The pointer position reported in the event is always the final position, not the initial position of the pointer. *root* is the root window for this position, and *root-x* and *root-y* are the pointer coordinates relative to the root's origin at the time of the event. *event* is the event window. If *event* is on the same screen as *root*, then *event-x* and

event-y are the pointer coordinates relative to the event window's origin. Otherwise, *event-x* and *event-y* are 0. In a *LeaveNotify* event, if a child of the event window contains the initial position of the pointer, then *child* is set to that child. Otherwise, it is *None*. For an *EnterNotify* event, if a child of the event window contains the final pointer position, then *child* is set to that child. Otherwise, it is *None*. If *event* is the focus window or an inferior of the focus window, then *focus* is *True*; otherwise, *focus* is *False*.

Normal pointer motion events have *mode=Normal*. Pseudo-motion events when a grab activates have *mode=Grab*, and pseudo-motion events when a grab deactivates have *mode=Ungrab*.

All *EnterNotify* and *LeaveNotify* events caused by a hierarchy change are generated after any hierarchy event caused by that change (that is, *UnmapNotify*, *MapNotify*, *ConfigureNotify*, *GravityNotify*, *CirculateNotify*), but the ordering of *EnterNotify* and *LeaveNotify* events with respect to *FocusOut*, *VisibilityNotify*, and *Expose* events is unspecified.

Normal events are generated as follows.

When the pointer moves from window A to window B and A is an inferior of B:

- *LeaveNotify* with *detail=Ancestor* is generated on A.
- *LeaveNotify* with *detail=Virtual* is generated on each window between A and B exclusive (in order).
- *EnterNotify* with *detail=Inferior* is generated on B.

When the pointer moves from window A to window B and B is an inferior of A:

- *LeaveNotify* with *detail=Inferior* is generated on A.
- *EnterNotify* with *detail=Virtual* is generated on each window between A and B exclusive (in order).
- *EnterNotify* with *detail=Ancestor* is generated on B.

When the pointer moves from window A to window B and window C is their closest common ancestor:

- *LeaveNotify* with *detail=Nonlinear* is generated on A.
- *LeaveNotify* with *detail=NonlinearVirtual* is generated on each window between A and C exclusive (in order).
- *EnterNotify* with *detail=NonlinearVirtual* is generated on each window between C and B exclusive (in order).
- *EnterNotify* with *detail=Nonlinear* is generated on B.

When the pointer moves from window A to window B on different screens:

- *LeaveNotify* with *detail=Nonlinear* is generated on A.
- If A is not a root window, *LeaveNotify* with *detail=NonlinearVirtual* is generated on each window above A up to and including its root (in order).
- If B is not a root window, *EnterNotify* with *detail=NonlinearVirtual* is generated on each window from B's root down to but not including B (in order).
- *EnterNotify* with *detail=Nonlinear* is generated on B.

When a pointer grab activates (but after any initial warp into a confine-to window and before generating any actual *ButtonPress* event that activates the grab), G is the grab-window for the grab, and P is the window the pointer is in:

- *EnterNotify* and *LeaveNotify* events with mode *Grab* are generated (as for *Normal* above) as if the pointer were to suddenly warp from its current position in P to some position in G. However, the pointer does not warp, and the pointer position is used as both the initial and final positions for the events.

  When a pointer grab deactivates (but after generating any actual *ButtonRelease* event that deactivates the grab), G is the grab-window for the grab, and P is the window the pointer is in:

- *EnterNotify* and *LeaveNotify* events with *mode=Ungrab* are generated (as for *Normal* above) as if the pointer were to suddenly warp from some position in G to its current position in P. However, the pointer does not warp, and the current pointer position is used as both the initial and final positions for the events.

Expose

window: WINDOW
x, *y*, *width*, *height*: CARD16
count: CARD16

This event is reported to clients selecting *Expose* on the window. It is generated when no valid contents are available for regions of a window, and either the regions are visible, the regions are viewable and the server is (perhaps newly) maintaining backing store on the window, or the window is not viewable but the server is (perhaps newly) honoring the window's *backing-store* attribute of *Always* or *WhenMapped*. The regions are decomposed into an unspecified set of rectangles, and an *Expose* event is generated for each rectangle.

For a given action causing exposure events, the set of events for a given window is guaranteed to be reported contiguously.

If *count* is 0, then no more *Expose* events for this window follow. If *count* is nonzero, then at least that many more *Expose* events for this window follow.

The *x* and *y* coordinates are relative to window's origin and specify the upper-left corner of a rectangle. *width* and *height* specify the extent of the rectangle.

Expose events are never generated on *InputOnly* windows.

All *Expose* events caused by a hierarchy change are generated after any hierarchy event caused by that change (for example, *UnmapNotify*, *MapNotify*, *ConfigureNotify*, *GravityNotify*, *CirculateNotify*). All *Expose* events on a given window are generated after any *VisibilityNotify* event on that window, but it is not required that all *Expose* events on all windows be generated after all *Visibility* events on all windows.

The ordering of *Expose* events with respect to *FocusOut*, *EnterNotify*, and *LeaveNotify* events is unspecified.

FocusIn, FocusOut

event: WINDOW
mode: {*Normal*, *WhileGrabbed*, *Grab*, *Ungrab*}
detail: {*Ancestor*, *Virtual*, *Inferior*, *Nonlinear*, *NonlinearVirtual*, *Pointer*, *PointerRoot*, *None*}

These events are generated when the input focus changes and are reported to clients selecting *FocusChange* on the window. Events generated by *SetInputFocus* when the keyboard is not grabbed have *mode=Normal*. Events generated by *SetInputFocus* when the keyboard is grabbed have *mode=WhileGrabbed*. Events generated when a keyboard grab activates have *mode=Grab*, and events generated when a keyboard grab deactivates have *mode=Ungrab*.

All *FocusOut* events caused by a window unmap are generated after any *UnmapNotify* event, but the ordering of *FocusOut* with respect to generated *EnterNotify*, *LeaveNotify*, *VisibilityNotify*, and *Expose* events is unspecified.

Normal and *WhileGrabbed* events are generated as follows.

When the focus moves from window A to window B, A is an inferior of B, and the pointer is in window P:

- *FocusOut* with *detail=Ancestor* is generated on A.
- *FocusOut* with *detail=Virtual* is generated on each window between A and B exclusive (in order).
- *FocusIn* with *detail=Inferior* is generated on B.
- If P is an inferior of B but P is not A or an inferior of A or an ancestor of A, *FocusIn* with *detail=Pointer* is generated on each window below B down to and including P (in order).

When the focus moves from window A to window B, B is an inferior of A, and the pointer is in window P:

- If P is an inferior of A but P is not an inferior of B or an ancestor of B, *FocusOut* with *detail=Pointer* is generated on each window from P up to but not including A (in order).
- *FocusOut* with *detail=Inferior* is generated on A.
- *FocusIn* with *detail=Virtual* is generated on each window between A and B exclusive (in order).
- *FocusIn* with *detail=Ancestor* is generated on B.

When the focus moves from window A to window B, window C is their closest common ancestor, and the pointer is in window P:

- If P is an inferior of A, *FocusOut* with *detail=Pointer* is generated on each window from P up to but not including A (in order).
- *FocusOut* with *detail=Nonlinear* is generated on A.
- *FocusOut* with *detail=NonlinearVirtual* is generated on each window between A and C exclusive (in order).
- *FocusIn* with *detail=NonlinearVirtual* is generated on each window between C and B exclusive (in order).
- *FocusIn* with *detail=Nonlinear* is generated on B.
- If P is an inferior of B, *FocusIn* with *detail=Pointer* is generated on each window below B down to and including P (in order).

When the focus moves from window A to window B on different screens and the pointer is in window P:

- If P is an inferior of A, *FocusOut* with *detail=Pointer* is generated on each window from P up to but not including A (in order).
- *FocusOut* with *detail=Nonlinear* is generated on A.
- If A is not a root window, *FocusOut* with *detail=NonlinearVirtual* is generated on each window above A up to and including its root (in order).
- If B is not a root window, *FocusIn* with *detail=NonlinearVirtual* is generated on each window from B's root down to but not including B (in order).

- *FocusIn* with *detail=Nonlinear* is generated on B.
- If P is an inferior of B, *FocusIn* with *detail=Pointer* is generated on each window below B down to and including P (in order).

When the focus moves from window A to *PointerRoot* (or *None*) and the pointer is in window P:

- If P is an inferior of A, *FocusOut* with *detail=Pointer* is generated on each window from P up to but not including A (in order).
- *FocusOut* with *detail=Nonlinear* is generated on A.
- If A is not a root window, *FocusOut* with *detail=NonlinearVirtual* is generated on each window above A up to and including its root (in order).
- *FocusIn* with *detail=PointerRoot* (or *None*) is generated on all root windows.
- If the new focus is *PointerRoot*, *FocusIn* with *detail=Pointer* is generated on each window from P's root down to and including P (in order).

When the focus moves from *PointerRoot* (or *None*) to window A and the pointer is in window P:

- If the old focus is *PointerRoot*, *FocusOut* with *detail=Pointer* is generated on each window from P up to and including P's root (in order).
- *FocusOut* with *detail=PointerRoot* (or *None*) is generated on all root windows.
- If A is not a root window, *FocusIn* with *detail=NonlinearVirtual* is generated on each window from A's root down to but not including A (in order).
- *FocusIn* with *detail=Nonlinear* is generated on A.
- If P is an inferior of A, *FocusIn* with *detail=Pointer* is generated on each window below A down to and including P (in order).

When the focus moves from *PointerRoot* to *None* (or *vice versa*) and the pointer is in window P:

- If the old focus is *PointerRoot*, *FocusOut* with *detail=Pointer* is generated on each window from P up to and including P's root (in order).
- *FocusOut* with *detail=PointerRoot* (or *None*) is generated on all root windows.
- *FocusIn* with *detail=None* (or *PointerRoot*) is generated on all root windows.
- If the new focus is *PointerRoot*, *FocusIn* with *detail=Pointer* is generated on each window from P's root down to and including P (in order).

When a keyboard grab activates (but before generating any actual *KeyPress* event that activates the grab), G is the grab-window for the grab, and F is the current focus:

- *FocusIn* and *FocusOut* events with *mode=Grab* are generated (as for *Normal* above) as if the focus were to change from F to G.

When a keyboard grab deactivates (but after generating any actual *KeyRelease* event that deactivates the grab), G is the grab-window for the grab, and F is the current focus:

- *FocusIn* and *FocusOut* events with *mode=Ungrab* are generated (as for *Normal* above) as if the focus were to change from G to F.

GraphicsExpose

drawable: DRAWABLE
x, *y*, *width*, *height*: CARD16
count: CARD16
major-opcode: CARD8
minor-opcode: CARD16

This event is reported to clients selecting *graphics-exposures* in a graphics context and is generated when a destination region could not be computed due to an obscured or out-of-bounds source region. All of the regions exposed by a given graphics request are reported contiguously. If *count* is 0, then no more *GraphicsExpose* events for this window follow. If *count* is nonzero, then at least that many more *GraphicsExpose* events for this window follow.

The *x* and *y* coordinates are relative to the drawable's origin and specify the upper-left corner of a rectangle. *width* and *height* specify the extent of the rectangle.

major-opcode and *minor-opcode* identify the graphics request used. For the core protocol, *major-opcode* is *CopyArea* or *CopyPlane*, and *minor-opcode* is 0.

GravityNotify

event, *window*: WINDOW
x, *y*: INT16

This event is reported to clients selecting *SubstructureNotify* on the parent and to clients selecting *StructureNotify* on the window. It is generated when a window is moved because of a change in size of the parent. *event* is the window on which the event was generated, and *window* is the window that is moved. The *x* and *y* coordinates are relative to the new parent's origin and specify the position of the upper-left outer corner of the window.

KeymapNotify

keys: LISTofCARD8

The value is a bit vector as described in *QueryKeymap*. This event is reported to clients selecting *KeymapState* on a window and is generated immediately after every *EnterNotify* and *FocusIn*.

KeyPress, KeyRelease

root, *event*: WINDOW
child: WINDOW or *None*
same-screen: BOOL
root-x, *root-y*, *event-x*, *event-y*: INT16
detail: KEYCODE
state: SETofKEYBUTMASK
time: TIMESTAMP

These events are generated when a key logically changes state. The generation of these logical changes may lag the physical changes if device event processing is frozen. These events are generated for all keys, even those mapped to modifier bits. The source of the event is the window the pointer is in. The *event* window is found by starting with the source window and looking up the hierarchy for the first window on which any client has selected interest in the event (provided no intervening window prohibits event generation by including the event type in its *do-not-propagate-mask*). The actual window used for reporting can be modified by active grabs and, in the case of keyboard events, can be modified by the focus window.

root is the root window of the source window, and *root-x* and *root-y* are the pointer coordinates relative to the origin of *root* at the time of the event. If *event* is on the same screen as root, then *event-x* and *event-y* are the pointer coordinates relative to the event window's origin. Otherwise, *event-x* and *event-y* are 0. If the source window is an inferior of the event window, then *child* is set to the child of the event window that is an ancestor of (or is) the source window. Otherwise, it is set to *None*. *state* gives the logical state of the buttons and modifier keys just before the event.

LeaveNotify

(See **EnterNotify**.)

MapNotify

event, *window*: WINDOW
override-redirect: BOOL

This event is reported to clients selecting *StructureNotify* on the window and to clients selecting *SubstructureNotify* on the parent. It is generated when the window changes state from unmapped to mapped. *event* is the window on which the event was generated, and *window* is the window that is mapped. *override-redirect* is from the window's attribute.

MappingNotify

request: {*Modifier*, *Keyboard*, *Pointer*}
first-keycode, *count*: CARD8

This event is sent to all clients. There is no mechanism to express disinterest in this event. *request* indicates the kind of change that occurred: *Modifiers* for a successful *SetModifierMapping*, *Keyboard* for a successful *ChangeKeyboardMapping*, and *Pointer* for a successful *SetPointerMapping*. If *request=Keyboard*, then *first-keycode* and *count* indicate the range of altered keycodes.

MapRequest

parent, *window*: WINDOW

This event is reported to the client selecting *SubstructureRedirect* on the parent and is generated when a *MapWindow* request is issued on an unmapped window with an *override-redirect* attribute of *False*.

MotionNotify

root, *event*: WINDOW
child: WINDOW or *None*
same-screen: BOOL
root-x, *root-y*, *event-x*, *event-y*: INT16
detail: {*Normal*, *Hint*}
state: SETofKEYBUTMASK
time: TIMESTAMP

These events are generated when the pointer logically moves. The generation of these logical changes may lag the physical changes if device event processing is frozen. The source of the event is the window the pointer is in. The *event* window is found by starting with the source window and looking up the hierarchy for the first window on which any client has selected interest in the event (provided no intervening window prohibits event generation by including the event type in its *do-not-propagate-mask*). The actual window used for reporting can be modified by active grabs and, in the case of keyboard events, can be modified by the focus

window.

root is the root window of the source window, and *root-x* and *root-y* are the pointer coordinates relative to root's origin at the time of the event. If *event* is on the same screen as *root*, then *event-x* and *event-y* are the pointer coordinates relative to the origin of *event*. Otherwise, *event-x* and *event-y* are 0. If the source window is an inferior of *event*, then *child* is set to the child of *event* that is an ancestor of (or is) the source window. Otherwise, it is set to *None*. *state* gives the logical state of the buttons and modifier keys just before the event.

MotionNotify events are only generated when the motion begins and ends in the window. The granularity of motion events is not guaranteed, but a client selecting for motion events is guaranteed to get at least one event when the pointer moves and comes to rest. Selecting *PointerMotion* receives events independent of the state of the pointer buttons. By selecting some subset of *Button[1-5]Motion* instead, *MotionNotify* events are received only when one or more of the specified buttons are pressed. By selecting *ButtonMotion*, *MotionNotify* events are received only when at least one button is pressed. The events are always of type *MotionNotify*, independent of the selection. If *PointerMotionHint* is selected, the server may send a single *MotionNotify* event with *detail=Hint* to the client for the event window until either the key or button state changes, the pointer leaves the event window, or the client issues a *QueryPointer* or *GetMotionEvents* request.

NoExpose

drawable: DRAWABLE
major-opcode: CARD8
minor-opcode: CARD16

This event is reported to clients selecting *graphics-exposures* in a graphics context and is generated when a graphics request that might produce *GraphicsExpose* events does not produce any. *drawable* specifies the destination used for the graphics request.

major-opcode and *minor-opcode* identify the graphics request used. For the core X protocol, *major-opcode* is *CopyArea* or *CopyPlane*, and *minor-opcode* is 0.

PropertyNotify

window: WINDOW
atom: ATOM
state: {*NewValue*, *Deleted*}
time: TIMESTAMP

This event is reported to clients selecting *PropertyChange* on *window* and is generated with *state=NewValue* when a property of *window* is changed using *ChangeProperty* or *RotateProperties*, even when adding zero-length data using *ChangeProperty* and when replacing all or part of a property with identical data using *ChangeProperty* or *RotateProperties*. It is generated with *state=Deleted* when a property of *window* is deleted using request *DeleteProperty* or *GetProperty*. *time* indicates the server time when the property was changed.

ReparentNotify

event, *window*, *parent*: WINDOW
x, *y*: INT16
override-redirect: BOOL

This event is reported to clients selecting *SubstructureNotify* on either the old or the new parent, and to clients selecting *StructureNotify* on the window. It is generated when *window* is reparented. *event* is the window on which the event was generated. *parent* specifies the new parent. The *x* and *y* coordinates are relative to the new parent's origin and specify the position of the upper-left outer corner of the window. The *override-redirect* flag is from the window's attribute.

ResizeRequest

window: WINDOW
width, *height*: CARD16

This event is reported to the client selecting *ResizeRedirect* on *window* and is generated when a *ConfigureWindow* request by some other client on *window* tries to change the size of *window*. *width* and *height* are the inside size, not including the border.

SelectionClear

owner: WINDOW
selection: ATOM
time: TIMESTAMP

This event is reported to the current owner of *selection* and is generated when a new owner is being defined by means of *SetSelectionOwner*. *time* is the *last-change-time* recorded for the selection. *owner* is the window that was specified by the current owner in its *SetSelectionOwner* request.

SelectionNotify

requestor: WINDOW
selection, *target*: ATOM
property: ATOM or *None*
time: TIMESTAMP or *CurrentTime*

This event is generated by the server in response to a *ConvertSelection* request when there is no owner for the selection. When there is an owner, it should be generated by the owner using *SendEvent*. The owner of a selection should send this event to a requestor either when a selection has been converted and stored as a property or when a selection conversion could not be performed (indicated with *property=None*).

SelectionRequest

owner: WINDOW
selection: ATOM
target: ATOM
property: ATOM or *None*
requestor: WINDOW
time: TIMESTAMP or *CurrentTime*

This event is reported to the owner of *selection* and is generated when a client issues a *ConvertSelection* request. *owner* is the window specified in the *SetSelectionOwner* request. The

remaining parameters are as in the *ConvertSelection* request.

The owner should convert *selection* based on the type of *target* and send a *SelectionNotify* back to the requestor.

UnmapNotify

event, *window*: WINDOW
from-configure: BOOL

This event is reported to clients selecting *StructureNotify* on *window* and to clients selecting *SubstructureNotify* on the parent. It is generated when *window* changes state from mapped to unmapped. *event* is the window on which the event was generated, and *window* is the window that is unmapped. The *from-configure* flag is *True* if the event was generated as a result of the window's parent being resized when the window itself had a *win-gravity* of *Unmap*.

VisibilityNotify

window: WINDOW
state: {*Unobscured*, *PartiallyObscured*, *FullyObscured*}

This event is reported to clients selecting *VisibilityChange* on *window*. In the following, the state of *window* is calculated ignoring all subwindows. When a window changes state from partially or fully obscured or not viewable to viewable and completely unobscured, an event with *Unobscured* is generated. When a window changes state from viewable and completely unobscured or not viewable, to viewable and partially obscured, an event with *PartiallyObscured* is generated. When a window changes state from viewable and completely unobscured, from viewable and partially obscured, or from not viewable to viewable and fully obscured, an event with *FullyObscured* is generated.

VisibilityNotify events are never generated on *InputOnly* windows.

All *VisibilityNotify* events caused by a hierarchy change are generated after any hierarchy event caused by that change (for example, *UnmapNotify*, *MapNotify*, *ConfigureNotify*, *GravityNotify*, *CirculateNotify*). Any *VisibilityNotify* event on a given window is generated before any *Expose* events on that window, but it is not required that all *VisibilityNotify* events on all windows be generated before all *Expose* events on all windows. The ordering of *VisibilityNotify* events with respect to *FocusOut*, *EnterNotify*, and *LeaveNotify* events is unspecified.

4.2 Button-Press Procedure

When a button press is processed with the pointer in some window W and no active pointer grab is in progress, the ancestors of W are searched from the root down, looking for a passive grab to activate. If no matching passive grab on the button exists, then an active grab is started automatically for the client receiving the event, and the *last-pointer-grab* time is set to the current server time.

The effect of the grab is essentially equivalent to a *GrabButton* with parameters:

Parameter	Value
event-window	Event window.
event-mask	Client's selected pointer events on the event window.
pointer-mode, keyboard-mode	*Asynchronous.*
owner-events	*True* if the client has *OwnerGrabButton* selected on the event window, otherwise *False.*
confine-to	*None.*
cursor	*None.*

The grab is terminated automatically when the logical state of the pointer has all buttons released. *UngrabPointer* and *ChangeActivePointerGrab* can both be used to modify the active grab.

Chapter 5

Protocol Encoding

For the syntactic conventions used in this chapter, see Section 1.3.3 on page 3.

5.1 Common Types

ARC

2	INT16	x
2	INT16	y
2	CARD16	width
2	CARD16	height
2	INT16	angle1
2	INT16	angle2

ATOM: CARD32

BITGRAVITY

0	Forget
1	NorthWest
2	North
3	NorthEast
4	West
5	Center
6	East
7	SouthWest
8	South
9	SouthEast
10	Static

BITMASK: CARD32

BOOL

0	False
1	True

BUTTON: CARD8

BYTE: 8-bit value

CARD8: 8-bit unsigned integer

CARD16: 16-bit unsigned integer

CARD32: 32-bit unsigned integer

CHAR2B

1	CARD8	byte1
1	CARD8	byte2

COLORMAP: CARD32

CURSOR: CARD32

DRAWABLE: CARD32

FONT: CARD32

FONTABLE: CARD32

GCONTEXT: CARD32

HOST

1			family
	0	Internet	
	1	DECnet	
	2	Chaos	
1			unused
2	n		length of address
n	LISTofBYTE		address
p			unused, p=pad(n)

INT8: 8-bit signed integer

INT16: 16-bit signed integer

INT32: 32-bit signed integer

KEYCODE: CARD8

KEYSYM: CARD32

LISTof*TYPE*

In this Technical Standard the LISTof notation strictly means some number of repetitions of an encoding; the actual length of the list is encoded elsewhere.

PIXMAP: CARD32

POINT

2	INT16	x
2	INT16	y

RECTANGLE

2	INT16	x
2	INT16	y
2	CARD16	width
2	CARD16	height

SETof*TYPE*

A set is always represented by a bitmask, with a 1-bit indicating presence in the set.

SETofDEVICEEVENT

#0x00000001	KeyPress
#0x00000002	KeyRelease
#0x00000004	ButtonPress
#0x00000008	ButtonRelease
#0x00000040	PointerMotion
#0x00000100	Button1Motion
#0x00000200	Button2Motion
#0x00000400	Button3Motion
#0x00000800	Button4Motion
#0x00001000	Button5Motion
#0x00002000	ButtonMotion
#0xFFFFC0B0	unused*

SETofEVENT

#0x00000001	KeyPress
#0x00000002	KeyRelease
#0x00000004	ButtonPress
#0x00000008	ButtonRelease
#0x00000010	EnterWindow
#0x00000020	LeaveWindow
#0x00000040	PointerMotion
#0x00000080	PointerMotionHint
#0x00000100	Button1Motion
#0x00000200	Button2Motion
#0x00000400	Button3Motion
#0x00000800	Button4Motion
#0x00001000	Button5Motion
#0x00002000	ButtonMotion
#0x00004000	KeymapState
#0x00008000	Expose
#0x00010000	VisibilityChange
#0x00020000	StructureNotify
#0x00040000	ResizeRedirect
#0x00080000	SubstructureNotify
#0x00100000	SubstructureRedirect
#0x00200000	FocusChange
#0x00400000	PropertyChange
#0x00800000	ColormapChange
#0x01000000	OwnerGrabButton
#0xFE000000	unused*

* **Application Usage:** Must be set to 0.

SETofKEYBUTMASK

#0x0001	Shift
#0x0002	Lock
#0x0004	Control
#0x0008	Mod1
#0x0010	Mod2
#0x0020	Mod3
#0x0040	Mod4
#0x0080	Mod5
#0x0100	Button1
#0x0200	Button2
#0x0400	Button3
#0x0800	Button4
#0x1000	Button5
#0xE000	unused*

SETofKEYMASK

#0x0001	Shift
#0x0002	Lock
#0x0004	Control
#0x0008	Mod1
#0x0010	Mod2
#0x0020	Mod3
#0x0040	Mod4
#0x0080	Mod5
#0xFF00	unused*

SETofPOINTEREVENT

#0x00000004	ButtonPress
#0x00000008	ButtonRelease
#0x00000010	EnterWindow
#0x00000020	LeaveWindow
#0x00000040	PointerMotion
#0x00000080	PointerMotionHint
#0x00000100	Button1Motion
#0x00000200	Button2Motion
#0x00000400	Button3Motion
#0x00000800	Button4Motion
#0x00001000	Button5Motion
#0x00002000	ButtonMotion
#0x00004000	KeymapState
#0xFFFF8003	unused*

STR

1	n	length of name in bytes
n	STRING8	name

STRING8: LISTofCARD8

STRING16: LISTofCHAR2B

TIMESTAMP: CARD32

VISUALID: CARD32

WINDOW: CARD32

WINGRAVITY

0	Unmap
1	NorthWest
2	North
3	NorthEast
4	West
5	Center
6	East
7	SouthWest
8	South
9	SouthEast
10	Static

5.2 Keyboards and Pointers

KEYCODE values are always greater than 7 (and less than 256).

KEYSYM values with the bit #0x10000000 set are reserved as implementation-dependent.

The names and encodings of the standard KEYSYM values are contained in Appendix A.

BUTTON values are numbered starting with 1.

5.3 Predefined Atoms

PRIMARY	1	WM_HINTS	35
SECONDARY	2	WM_CLIENT_MACHINE	36
ARC	3	WM_ICON_NAME	37
ATOM	4	WM_ICON_SIZE	38
BITMAP	5	WM_NAME	39
CARDINAL	6	WM_NORMAL_HINTS	40
COLORMAP	7	WM_SIZE_HINTS	41
CURSOR	8	WM_ZOOM_HINTS	42
CUT_BUFFER0	9	MIN_SPACE	43
CUT_BUFFER1	10	NORM_SPACE	44
CUT_BUFFER2	11	MAX_SPACE	45
CUT_BUFFER3	12	END_SPACE	46
CUT_BUFFER4	13	SUPERSCRIPT_X	47
CUT_BUFFER5	14	SUPERSCRIPT_Y	48
CUT_BUFFER6	15	SUBSCRIPT_X	49
CUT_BUFFER7	16	SUBSCRIPT_Y	50
DRAWABLE	17	UNDERLINE_POSITION	51
FONT	18	UNDERLINE_THICKNESS	52
INTEGER	19	STRIKEOUT_ASCENT	53
PIXMAP	20	STRIKEOUT_DESCENT	54
POINT	21	ITALIC_ANGLE	55
RECTANGLE	22	X_HEIGHT	56
RESOURCE_MANAGER	23	QUAD_WIDTH	57
RGB_COLOR_MAP	24	WEIGHT	58
RGB_BEST_MAP	25	POINT_SIZE	59
RGB_BLUE_MAP	26	RESOLUTION	60
RGB_DEFAULT_MAP	27	COPYRIGHT	61
RGB_GRAY_MAP	28	NOTICE	62
RGB_GREEN_MAP	29	FONT_NAME	63
RGB_RED_MAP	30	FAMILY_NAME	64
STRING	31	FULL_NAME	65
VISUALID	32	CAP_HEIGHT	66
WINDOW	33	WM_CLASS	67
WM_COMMAND	34	WM_TRANSIENT_FOR	68

5.4 Connection Setup

For TCP connections, displays on a given host are numbered starting from 0, and the server for display N listens and accepts connections on port 6000 + N.

Information sent by the client at connection setup:

1			byte-order
	#0x42	MSB first	
	#0x6C	LSB first	
1			unused
2	CARD16		protocol-major-version
2	CARD16		protocol-minor-version
2	n		length of authorization-protocol-name
2	d		length of authorization-protocol-data
2			unused
n	STRING8		authorization-protocol-name
p			unused, p=pad(n)
d	STRING8		authorization-protocol-data
q			unused, q=pad(d)

Except where explicitly noted in the protocol, all 16-bit and 32-bit quantities sent by the client are transmitted with the specified byte order, and all 16-bit and 32-bit quantities returned by the server are transmitted with this byte order.

Information received by the client if the connection is refused:

1	0	Failed
1	n	length of reason in bytes
2	CARD16	protocol-major-version
2	CARD16	protocol-minor-version
2	(n+p)/4	length in 4-byte units of ‘‘additional data’’
n	STRING8	reason
p		unused, p=pad(n)

Information received by the client if further authentication is required:

1	2	Authenticate
5		unused
2	(n+p)/4	length in 4-byte units of ‘‘additional data’’
n	STRING8	reason
p		unused, p=pad(n)

Information received by the client if authorization is accepted:

1	1		Success
1			unused
2	CARD16		protocol-major-version
2	CARD16		protocol-minor-version
2	8+2n+(v+p+m)/4		length in 4-byte units of ‘‘additional data’’
4	CARD32		release-number
4	CARD32		resource-id-base
4	CARD32		resource-id-mask
4	CARD32		motion-buffer-size
2	v		length of vendor
2	CARD16		maximum-request-length
1	CARD8		number of SCREENs in roots
1	n		number for FORMATs in pixmap-formats
1			image-byte-order
	0	LSBFirst	
	1	MSBFirst	
1			bitmap-format-bit-order
	0	LeastSignificant	
	1	MostSignificant	
1	CARD8		bitmap-format-scanline-unit
1	CARD8		bitmap-format-scanline-pad
1	KEYCODE		min-keycode
1	KEYCODE		max-keycode
4			unused
v	STRING8		vendor
p			unused, p=pad(v)
8n	LISTofFORMAT		pixmap-formats
m	LISTofSCREEN		roots (m is always a multiple of 4)

FORMAT

1	CARD8	depth
1	CARD8	bits-per-pixel
1	CARD8	scanline-pad
5		unused

SCREEN

4	WINDOW		root
4	COLORMAP		default-colormap
4	CARD32		white-pixel
4	CARD32		black-pixel
4	SETofEVENT		current-input-masks
2	CARD16		width-in-pixels
2	CARD16		height-in-pixels
2	CARD16		width-in-millimeters
2	CARD16		height-in-millimeters
2	CARD16		min-installed-maps
2	CARD16		max-installed-maps
4	VISUALID		root-visual
1			backing-store
	0	Never	
	1	WhenMapped	
	2	Always	
1	BOOL		save-unders
1	CARD8		root-depth
1	CARD8		number of DEPTHs in allowed-depths
n	LISTofDEPTH		allowed-depths (n is always a multiple of 4)

DEPTH

1	CARD8		depth
1			unused
2	n		number of VISUALTYPES in visuals
4			unused
24n	LISTofVISUALTYPE		visuals

VISUALTYPE

4	VISUALID		visual-id
1			class
	0	StaticGray	
	1	GrayScale	
	2	StaticColor	
	3	PseudoColor	
	4	TrueColor	
	5	DirectColor	
1	CARD8		bits-per-rgb-value
2	CARD16		colormap-entries
4	CARD32		red-mask
4	CARD32		green-mask
4	CARD32		blue-mask
4			unused

5.5 Requests

This section presents each request in the core X protocol, in order of the request opcode.

CreateWindow

1	1		request opcode
1	CARD8		depth
2	8+n		request length
4	WINDOW		wid
4	WINDOW		parent
2	INT16		x
2	INT16		y
2	CARD16		width
2	CARD16		height
2	CARD16		border-width
2			class
	0	CopyFromParent	
	1	InputOutput	
	2	InputOnly	
4	VISUALID		visual
	0	CopyFromParent	
4	BITMASK		value-mask (has n bits set to 1)
	#0x00000001	background-pixmap	
	#0x00000002	background-pixel	
	#0x00000004	border-pixmap	
	#0x00000008	border-pixel	
	#0x00000010	bit-gravity	
	#0x00000020	win-gravity	
	#0x00000040	backing-store	
	#0x00000080	backing-planes	
	#0x00000100	backing-pixel	
	#0x00000200	override-redirect	
	#0x00000400	save-under	
	#0x00000800	event-mask	
	#0x00001000	do-not-propagate-mask	
	#0x00002000	colormap	
	#0x00004000	cursor	
4n	LISTofVALUE		value-list

VALUE

4	PIXMAP		background-pixmap
	0	None	
	1	ParentRelative	
4	CARD32		background-pixel
4	PIXMAP		border-pixmap
	0	CopyFromParent	
4	CARD32		border-pixel
1	BITGRAVITY		bit-gravity
1	WINGRAVITY		win-gravity
1			backing-store
	0	NotUseful	
	1	WhenMapped	
	2	Always	
4	CARD32		backing-planes
4	CARD32		backing-pixel
1	BOOL		override-redirect
1	BOOL		save-under
4	SETofEVENT		event-mask
4	SETofDEVICEEVENT		do-not-propagate-mask
4	COLORMAP		colormap
	0	CopyFromParent	
4	CURSOR		cursor
	0	None	

ChangeWindowAttributes

1	2	request opcode
1		unused
2	3+n	request length
4	WINDOW	window
4	BITMASK encodings are the same as for *CreateWindow*	value-mask (has n bits set to 1)
4n	LISTofVALUE encodings are the same as for *CreateWindow*	value-list

GetWindowAttributes

1	3	request opcode
1		unused
2	2	request length
4	WINDOW	window

→

1	1		Reply
1			backing-store
	0	NotUseful	
	1	WhenMapped	
	2	Always	
2	CARD16		sequence number
4	3		reply length
4	VISUALID		visual
2			class
	1	InputOutput	
	2	InputOnly	
1	BITGRAVITY		bit-gravity
1	WINGRAVITY		win-gravity
4	CARD32		backing-planes
4	CARD32		backing-pixel
1	BOOL		save-under
1	BOOL		map-is-installed
1			map-state
	0	Unmapped	
	1	Unviewable	
	2	Viewable	
1	BOOL		override-redirect
4	COLORMAP		colormap
	0	None	
4	SETofEVENT		all-event-masks
4	SETofEVENT		your-event-mask
2	SETofDEVICEEVENT		do-not-propagate-mask
2			unused

DestroyWindow

1	4		request opcode
1			unused
2	2		request length
4	WINDOW		window

DestroySubwindows

1	5		request opcode
1			unused
2	2		request length
4	WINDOW		window

ChangeSaveSet

1	6		request opcode
1			mode
	0	Insert	
	1	Delete	
2	2		request length
4	WINDOW		window

ReparentWindow

1	7	request opcode
1		unused
2	4	request length
4	WINDOW	window
4	WINDOW	parent
2	INT16	x
2	INT16	y

MapWindow

1	8	request opcode
1		unused
2	2	request length
4	WINDOW	window

MapSubwindows

1	9	request opcode
1		unused
2	2	request length
4	WINDOW	window

UnmapWindow

1	10	request opcode
1		unused
2	2	request length
4	WINDOW	window

UnmapSubwindows

1	11	request opcode
1		unused
2	2	request length
4	WINDOW	window

ConfigureWindow

1	12		request opcode
1			unused
2	3+n		request length
4	WINDOW		window
2	BITMASK		value-mask (has n bits set to 1)
	#0x0001	x	
	#0x0002	y	
	#0x0004	width	
	#0x0008	height	
	#0x0010	border-width	
	#0x0020	sibling	
	#0x0040	stack-mode	
2			unused
4n	LISTofVALUE		value-list

VALUE

2	INT16		x
2	INT16		y
2	CARD16		width
2	CARD16		height
2	CARD16		border-width
4	WINDOW		sibling
1			stack-mode
	0	Above	
	1	Below	
	2	TopIf	
	3	BottomIf	
	4	Opposite	

CirculateWindow

1	13		request opcode
1			direction
	0	RaiseLowest	
	1	LowerHighest	
2	2		request length
4	WINDOW		window

GetGeometry

1	14		request opcode
1			unused
2	2		request length
4	DRAWABLE		drawable

→

1	1		Reply
1	CARD8		depth
2	CARD16		sequence number
4	0		reply length
4	WINDOW		root
2	INT16		x
2	INT16		y
2	CARD16		width
2	CARD16		height
2	CARD16		border-width
10			unused

QueryTree

1	15		request opcode
1			unused
2	2		request length
4	WINDOW		window

→

1	1		Reply
1			unused
2	CARD16		sequence number
4	n		reply length
4	WINDOW		root
4	WINDOW		parent
	0	None	
2	n		number of WINDOWs in children
14			unused
4n	LISTofWINDOW		children

InternAtom

1	16		request opcode
1	BOOL		only-if-exists
2	2+(n+p)/4		request length
2	n		length of name
2			unused
n	STRING8		name
p			unused, p=pad(n)

→

1	1		Reply
1			unused
2	CARD16		sequence number
4	0		reply length
4	ATOM		atom
	0	None	
20			unused

GetAtomName

1	17		request opcode
1			unused
2	2		request length
4	ATOM		atom

→

1	1		Reply
1			unused
2	CARD16		sequence number
4	(n+p)/4		reply length
2	n		length of name
22			unused
n	STRING8		name
p			unused, p=pad(n)

ChangeProperty

1	18		request opcode
1			mode
	0	Replace	
	1	Prepend	
	2	Append	
2	6+(n+p)/4		request length
4	WINDOW		window
4	ATOM		property
4	ATOM		type
1	CARD8		format
3			unused
4	CARD32		length of data in format units (n for format = 8) (n/2 for format = 16) (n/4 for format = 32)
n	LISTofBYTE		data (n is a multiple of 2 for format = 16) (n is a multiple of 4 for format = 32)
p			unused, p=pad(n)

DeleteProperty

1	19	request opcode
1		unused
2	3	request length
4	WINDOW	window
4	ATOM	property

GetProperty

1	20		request opcode
1	BOOL		delete
2	6		request length
4	WINDOW		window
4	ATOM		property
4	ATOM		type
	0	AnyPropertyType	
4	CARD32		long-offset
4	CARD32		long-length

→

1	1		Reply
1	CARD8		format
2	CARD16		sequence number
4	(n+p)/4		reply length
4	ATOM		type
	0	None	
4	CARD32		bytes-after
4	CARD32		length of value in format units (0 for format = 0) (n for format = 8) (n / 2 for format = 16) (n / 4 for format = 32)
12			unused
n	LISTofBYTE		value (n is 0 for format = 0) (n is a multiple of 2 for format = 16) (n is a multiple of 4 for format = 32)
p			unused, p=pad(n)

ListProperties

1	21		request opcode
1			unused
2	2		request length
4	WINDOW		window

→

1	1		Reply
1			unused
2	CARD16		sequence number
4	n		reply length
2	n		number of ATOMs in atoms
22			unused
4n	LISTofATOM		atoms

SetSelectionOwner

1	22		request opcode
1			unused
2	4		request length
4	WINDOW		owner
	0	None	
4	ATOM		selection
4	TIMESTAMP		time
	0	CurrentTime	

GetSelectionOwner

1	23		request opcode
1			unused
2	2		request length
4	ATOM		selection

→

1	1		Reply
1			unused
2	CARD16		sequence number
4	0		reply length
4	WINDOW		owner
	0	None	
20			unused

ConvertSelection

1	24		request opcode
1			unused
2	6		request length
4	WINDOW		requestor
4	ATOM		selection
4	ATOM		target
4	ATOM		property
	0	None	
4	TIMESTAMP		time
	0	CurrentTime	

SendEvent

1	25		request opcode
1	BOOL		propagate
2	11		request length
4	WINDOW		destination
	0	PointerWindow	
	1	InputFocus	
4	SETofEVENT		event-mask
32	standard event format†	event	

GrabPointer

1	26		request opcode
1	BOOL		owner-events
2	6		request length
4	WINDOW		grab-window
2	SETofPOINTEREVENT		event-mask
1			pointer-mode
	0	Synchronous	
	1	Asynchronous	
1			keyboard-mode
	0	Synchronous	
	1	Asynchronous	
4	WINDOW		confine-to
	0	None	
4	CURSOR		cursor
	0	None	
4	TIMESTAMP		time
	0	CurrentTime	

→

1	1		Reply
1			status
	0	Success	
	1	AlreadyGrabbed	
	2	InvalidTime	
	3	NotViewable	
	4	Frozen	
2	CARD16		sequence number
4	0		reply length
24			unused

UngrabPointer

1	27		request opcode
1			unused
2	2		request length
4	TIMESTAMP		time
	0	CurrentTime	

† See Section 5.6 on page 147.

GrabButton

1	28		request opcode
1	BOOL		owner-events
2	6		request length
4	WINDOW		grab-window
2	SETofPOINTEREVENT		event-mask
1			pointer-mode
	0	Synchronous	
	1	Asynchronous	
1			keyboard-mode
	0	Synchronous	
	1	Asynchronous	
4	WINDOW		confine-to
	0	None	
4	CURSOR		cursor
	0	None	
1	BUTTON		button
	0	AnyButton	
1			unused
2	SETofKEYMASK		modifiers
	#0x8000	AnyModifier	

UngrabButton

1	29		request opcode
1	BUTTON		button
	0	AnyButton	
2	3		request length
4	WINDOW		grab-window
2	SETofKEYMASK		modifiers
	#0x8000	AnyModifier	
2			unused

ChangeActivePointerGrab

1	30		request opcode
1			unused
2	4		request length
4	CURSOR		cursor
	0	None	
4	TIMESTAMP		time
	0	CurrentTime	
2	SETofPOINTEREVENT		event-mask
2			unused

GrabKeyboard

1	31		request opcode
1	BOOL		owner-events
2	4		request length
4	WINDOW		grab-window
4	TIMESTAMP		time
	0	CurrentTime	
1			pointer-mode
	0	Synchronous	
	1	Asynchronous	
1			keyboard-mode
	0	Synchronous	
	1	Asynchronous	
2			unused

→

1	1		Reply
1			status
	0	Success	
	1	AlreadyGrabbed	
	2	InvalidTime	
	3	NotViewable	
	4	Frozen	
2	CARD16		sequence number
4	0		reply length
24			unused

UngrabKeyboard

1	32		request opcode
1			unused
2	2		request length
4	TIMESTAMP		time
	0	CurrentTime	

GrabKey

1	33		request opcode
1	BOOL		owner-events
2	4		request length
4	WINDOW		grab-window
2	SETofKEYMASK		modifiers
	#0x8000	AnyModifier	
1	KEYCODE		key
	0	AnyKey	
1			pointer-mode
	0	Synchronous	
	1	Asynchronous	
1			keyboard-mode
	0	Synchronous	
	1	Asynchronous	
3			unused

UngrabKey

1	34		request opcode
1	KEYCODE		key
	0	AnyKey	
2	3		request length
4	WINDOW		grab-window
2	SETofKEYMASK		modifiers
	#0x8000	AnyModifier	
2			unused

AllowEvents

1	35		request opcode
1			mode
	0	AsyncPointer	
	1	SyncPointer	
	2	ReplayPointer	
	3	AsyncKeyboard	
	4	SyncKeyboard	
	5	ReplayKeyboard	
	6	AsyncBoth	
	7	SyncBoth	
2	2		request length
4	TIMESTAMP		time
	0	CurrentTime	

GrabServer

1	36		request opcode
1			unused
2	1		request length

UngrabServer

1	37		request opcode
1			unused
2	1		request length

QueryPointer

1	38		request opcode
1			unused
2	2		request length
4	WINDOW		window

→

1	1		Reply
1	BOOL		same-screen
2	CARD16		sequence number
4	0		reply length
4	WINDOW		root
4	WINDOW		child
	0	None	
2	INT16		root-x
2	INT16		root-y
2	INT16		win-x
2	INT16		win-y
2	SETofKEYBUTMASK		mask
6			unused

GetMotionEvents

1	39		request opcode
1			unused
2	4		request length
4	WINDOW		window
4	TIMESTAMP		start
	0	CurrentTime	
4	TIMESTAMP		stop
	0	CurrentTime	

→

1	1		Reply
1			unused
2	CARD16		sequence number
4	2n		reply length
4	n		number of TIMECOORDs in events
20			unused
8n	LISTofTIMECOORD		events

TIMECOORD

4	TIMESTAMP		time
2	INT16		x
2	INT16		y

TranslateCoordinates

1	40		request opcode
1			unused
2	4		request length
4	WINDOW		src-window
4	WINDOW		dst-window
2	INT16		src-x
2	INT16		src-y

→

1	1		Reply
1	BOOL		same-screen
2	CARD16		sequence number
4	0		reply length
4	WINDOW		child
	0	None	
2	INT16		dst-x
2	INT16		dst-y
16			unused

WarpPointer

1	41		request opcode
1			unused
2	6		request length
4	WINDOW		src-window
	0	None	
4	WINDOW		dst-window
	0	None	
2	INT16		src-x
2	INT16		src-y
2	CARD16		src-width
2	CARD16		src-height
2	INT16		dst-x
2	INT16		dst-y

SetInputFocus

1	42		request opcode
1			revert-to
	0	None	
	1	PointerRoot	
	2	Parent	
2	3		request length
4	WINDOW		focus
	0	None	
	1	PointerRoot	
4	TIMESTAMP		time
	0	CurrentTime	

GetInputFocus

1	43		request opcode
1			unused
2	1		request length

→

1	1		Reply
1			revert-to
	0	None	
	1	PointerRoot	
	2	Parent	
2	CARD16		sequence number
4	0		reply length
4	WINDOW		focus
	0	None	
	1	PointerRoot	
20			unused

QueryKeymap

1	44		request opcode
1			unused
2	1		request length

→

1	1		Reply
1			unused
2	CARD16		sequence number
4	2		reply length
32	LISTofCARD8		keys

OpenFont

1	45		request opcode
1			unused
2	3+(n+p)/4		request length
4	FONT		fid
2	n		length of name
2			unused
n	STRING8		name
p			unused, p=pad(n)

CloseFont

1	46		request opcode
1			unused
2	2		request length
4	FONT		font

QueryFont

1	47		request opcode
1			unused
2	2		request length
4	FONTABLE		font (or graphics context)

→

1	1		Reply
1			unused
2	CARD16		sequence number
4	7+2n+3m		reply length
12	CHARINFO		min-bounds
4			unused
12	CHARINFO		max-bounds
4			unused
2	CARD16		min-char-or-byte2
2	CARD16		max-char-or-byte2
2	CARD16		default-char
2	n		number of FONTPROPs in properties
1			draw-direction
	0	LeftToRight	
	1	RightToLeft	
1	CARD8		min-byte1
1	CARD8		max-byte1
1	BOOL		all-chars-exist
2	INT16		font-ascent
2	INT16		font-descent
4	m		number of CHARINFOs in char-infos
8n	LISTofFONTPROP		properties
12m	LISTofCHARINFO		char-infos

FONTPROP

4	ATOM		name
4	<32-bits>		value

CHARINFO

2	INT16		left-side-bearing
2	INT16		right-side-bearing
2	INT16		character-width
2	INT16		ascent
2	INT16		descent
2	CARD16		attributes

QueryTextExtents

1	48		request opcode
1	BOOL		odd length, True if p = 2
2	2+(2n+p)/4		request length
4	FONTABLE		font (or graphics context)
2n	STRING16		string
p			unused, p=pad(2n)

→

1	1		Reply
1			draw-direction
	0	LeftToRight	
	1	RightToLeft	
2	CARD16		sequence number
4	0		reply length
2	INT16		font-ascent
2	INT16		font-descent
2	INT16		overall-ascent
2	INT16		overall-descent
4	INT32		overall-width
4	INT32		overall-left
4	INT32		overall-right
4			unused

ListFonts

1	49		request opcode
1			unused
2	2+(n+p)/4		request length
2	CARD16		max-names
2	n		length of pattern
n	STRING8		pattern
p			unused, p=pad(n)

→

1	1		Reply
1			unused
2	CARD16		sequence number
4	(n+p)/4		reply length
2	CARD16		number of STRs in names
22			unused
n	LISTofSTR		names
p			unused, p=pad(n)

ListFontsWithInfo

1	50		request opcode
1			unused
2	2+(n+p)/4		request length
2	CARD16		max-names
2	n		length of pattern
n	STRING8		pattern
p			unused, p=pad(n)

→ (except for last in series)

1	1		Reply
1	n		length of name in bytes
2	CARD16		sequence number
4	7+2m+(n+p)/4		reply length
12	CHARINFO		min-bounds
4			unused
12	CHARINFO		max-bounds
4			unused
2	CARD16		min-char-or-byte2
2	CARD16		max-char-or-byte2
2	CARD16		default-char
2	m		number of FONTPROPs in properties
1			draw-direction
	0	LeftToRight	
	1	RightToLeft	
1	CARD8		min-byte1
1	CARD8		max-byte1
1	BOOL		all-chars-exist
2	INT16		font-ascent
2	INT16		font-descent
4	CARD32		replies-hint
8m	LISTofFONTPROP		properties
n	STRING8		name
p			unused, p=pad(n)

FONTPROP
Encodings are the same as for *QueryFont*.

CHARINFO
Encodings are the same as for *QueryFont*.

→ (last in series)

1	1	Reply
1	0	last-reply indicator
2	CARD16	sequence number
4	7	reply length
52		unused

SetFontPath

1	51	request opcode
1		unused
2	2+(n+p)/4	request length
2	CARD16	number of STRs in path
2		unused
n	LISTofSTR	path
p		unused, p=pad(n)

GetFontPath

1	52	request opcode
1		unused
2	1	request list

→

1	1	Reply
1		unused
2	CARD16	sequence number
4	(n+p)/4	reply length
2	CARD16	number of STRs in path
22		unused
n	LISTofSTR	path
p		unused, p=pad(n)

CreatePixmap

1	53	request opcode
1	CARD8	depth
2	4	request length
4	PIXMAP	pid
4	DRAWABLE	drawable
2	CARD16	width
2	CARD16	height

FreePixmap

1	54	request opcode
1		unused
2	2	request length
4	PIXMAP	pixmap

CreateGC

1	55		request opcode
1			unused
2	4+n		request length
4	GCONTEXT		cid
4	DRAWABLE		drawable
4	BITMASK		value-mask (has n bits set to 1)
	#0x00000001	function	
	#0x00000002	plane-mask	
	#0x00000004	foreground	
	#0x00000008	background	
	#0x00000010	line-width	
	#0x00000020	line-style	
	#0x00000040	cap-style	
	#0x00000080	join-style	
	#0x00000100	fill-style	
	#0x00000200	fill-rule	
	#0x00000400	tile	
	#0x00000800	stipple	
	#0x00001000	tile-stipple-x-origin	
	#0x00002000	tile-stipple-y-origin	
	#0x00004000	font	
	#0x00008000	subwindow-mode	
	#0x00010000	graphics-exposures	
	#0x00020000	clip-x-origin	
	#0x00040000	clip-y-origin	
	#0x00080000	clip-mask	
	#0x00100000	dash-offset	
	#0x00200000	dashes	
	#0x00400000	arc-mode	
4n	LISTofVALUE		value-list

VALUE

1			function
	0	Clear	
	1	And	
	2	AndReverse	
	3	Copy	
	4	AndInverted	
	5	NoOp	
	6	Xor	
	7	Or	
	8	Nor	
	9	Equiv	
	10	Invert	
	11	OrReverse	
	12	CopyInverted	
	13	OrInverted	
	14	Nand	
	15	Set	
4	CARD32		plane-mask
4	CARD32		foreground
4	CARD32		background
2	CARD16		line-width
1			line-style
	0	Solid	
	1	OnOffDash	
	2	DoubleDash	
1			cap-style
	0	NotLast	
	1	Butt	
	2	Round	
	3	Projecting	
1			join-style
	0	Miter	
	1	Round	
	2	Bevel	
1			fill-style
	0	Solid	
	1	Tiled	
	2	Stippled	
	3	OpaqueStippled	
1			fill-rule
	0	EvenOdd	
	1	Winding	
4	PIXMAP		tile
4	PIXMAP		stipple
2	INT16		tile-stipple-x-origin
2	INT16		tile-stipple-y-origin
4	FONT		font
1			subwindow-mode
	0	ClipByChildren	
	1	IncludeInferiors	
1	BOOL		graphics-exposures

2	INT16		clip-x-origin
2	INT16		clip-y-origin
4	PIXMAP		clip-mask
	0	None	
2	CARD16		dash-offset
1	CARD8		dashes
1			arc-mode
	0	Chord	
	1	PieSlice	

ChangeGC

1	56	request opcode
1		unused
2	3+n	request length
4	GCONTEXT	gc
4	BITMASK encodings are the same as for *CreateGC*	value-mask (has n bits set to 1)
4n	LISTofVALUE encodings are the same as for *CreateGC*	value-list

CopyGC

1	57	request opcode
1		unused
2	4	request length
4	GCONTEXT	src-gc
4	GCONTEXT	dst-gc
4	BITMASK encodings are the same as for *CreateGC*	value-mask

SetDashes

1	58	request opcode
1		unused
2	3+(n+p)/4	request length
4	GCONTEXT	gc
2	CARD16	dash-offset
2	n	length of dashes
n	LISTofCARD8	dashes
p		unused, p=pad(n)

SetClipRectangles

1	59		request opcode
1			ordering
	0	UnSorted	
	1	YSorted	
	2	YXSorted	
	3	YXBanded	
2	3+2n		request length
4	GCONTEXT		gc
2	INT16		clip-x-origin
2	INT16		clip-y-origin
8n	LISTofRECTANGLE		rectangles

FreeGC

1	60	request opcode
1		unused
2	2	request length
4	GCONTEXT	gc

ClearArea

1	61	request opcode
1	BOOL	exposures
2	4	request length
4	WINDOW	window
2	INT16	x
2	INT16	y
2	CARD16	width
2	CARD16	height

CopyArea

1	62	request opcode
1		unused
2	7	request length
4	DRAWABLE	src-drawable
4	DRAWABLE	dst-drawable
4	GCONTEXT	gc
2	INT16	src-x
2	INT16	src-y
2	INT16	dst-x
2	INT16	dst-y
2	CARD16	width
2	CARD16	height

CopyPlane

1	63		request opcode
1			unused
2	8		request length
4	DRAWABLE		src-drawable
4	DRAWABLE		dst-drawable
4	GCONTEXT		gc
2	INT16		src-x
2	INT16		src-y
2	INT16		dst-x
2	INT16		dst-y
2	CARD16		width
2	CARD16		height
4	CARD32		bit-plane

PolyPoint

1	64		request opcode
1			coordinate-mode
	0	Origin	
	1	Previous	
2	3+n		request length
4	DRAWABLE		drawable
4	GCONTEXT		gc
4n	LISTofPOINT		points

PolyLine

1	65		request opcode
1			coordinate-mode
	0	Origin	
	1	Previous	
2	3+n		request length
4	DRAWABLE		drawable
4	GCONTEXT		gc
4n	LISTofPOINT		points

PolySegment

1	66	request opcode
1		unused
2	3+2n	request length
4	DRAWABLE	drawable
4	GCONTEXT	gc
8n	LISTofSEGMENT	segments

SEGMENT

2	INT16	x1
2	INT16	y1
2	INT16	x2
2	INT16	y2

PolyRectangle

1	67		request opcode
1			unused
2	3+2n		request length
4	DRAWABLE		drawable
4	GCONTEXT		gc
8n	LISTofRECTANGLE		rectangles

PolyArc

1	68		request opcode
1			unused
2	3+3n		request length
4	DRAWABLE		drawable
4	GCONTEXT		gc
12n	LISTofARC		arcs

FillPoly

1	69		request opcode
1			unused
2	4+n		request length
4	DRAWABLE		drawable
4	GCONTEXT		gc
1			shape
	0	Complex	
	1	Nonconvex	
	2	Convex	
1			coordinate-mode
	0	Origin	
	1	Previous	
2			unused
4n	LISTofPOINT		points

PolyFillRectangle

1	70		request opcode
1			unused
2	3+2n		request length
4	DRAWABLE		drawable
4	GCONTEXT		gc
8n	LISTofRECTANGLE		rectangles

PolyFillArc

1	71		request opcode
1			unused
2	3+3n		request length
4	DRAWABLE		drawable
4	GCONTEXT		gc
12n	LISTofARC		arcs

PutImage

1	72		request opcode
1			format
	0	Bitmap	
	1	XYPixmap	
	2	ZPixmap	
2	6+(n+p)/4		request length
4	DRAWABLE		drawable
4	GCONTEXT		gc
2	CARD16		width
2	CARD16		height
2	INT16		dst-x
2	INT16		dst-y
1	CARD8		left-pad
1	CARD8		depth
2			unused
n	LISTofBYTE		data
p			unused, p=pad(n)

GetImage

1	73		request opcode
1			format
	1	XYPixmap	
	2	ZPixmap	
2	5		request length
4	DRAWABLE		drawable
2	INT16		x
2	INT16		y
2	CARD16		width
2	CARD16		height
4	CARD32		plane-mask

→

1	1		Reply
1	CARD8		depth
2	CARD16		sequence number
4	(n+p)/4		reply length
4	VISUALID		visual
	0	None	
20			unused
n	LISTofBYTE		data
p			unused, p=pad(n)

PolyText8

1	74	request opcode
1		unused
2	4+(n+p)/4	request length
4	DRAWABLE	drawable
4	GCONTEXT	gc
2	INT16	x
2	INT16	y
n	LISTofTEXTITEM8	items
p		unused, p=pad(n) (p is always 0 or 1)

TEXTITEM8

1	m	length of string (cannot be 255)
1	INT8	delta
m	STRING8	string

or:

1	255	font-shift indicator
1		font byte 3 (most-significant)
1		font byte 2
1		font byte 1
1		font byte 0 (least-significant)

PolyText16

1	75	request opcode
1		unused
2	4+(n+p)/4	request length
4	DRAWABLE	drawable
4	GCONTEXT	gc
2	INT16	x
2	INT16	y
n	LISTofTEXTITEM16	items
p		unused, p=pad(n) (p must be 0 or 1)

TEXTITEM16

1	m	number of CHAR2Bs in string (cannot be 255)
1	INT8	delta
2m	STRING16	string

or:

1	255		font-shift indicator
1			font byte 3 (most-significant)
1			font byte 2
1			font byte 1
1			font byte 0 (least-significant)

ImageText8

1	76		request opcode
1	n		length of string
2	4+(n+p)/4		request length
4	DRAWABLE		drawable
4	GCONTEXT		gc
2	INT16		x
2	INT16		y
n	STRING8		string
p			unused, p=pad(n)

ImageText16

1	77		request opcode
1	n		number of CHAR2Bs in string
2	4+(2n+p)/4		request length
4	DRAWABLE		drawable
4	GCONTEXT		gc
2	INT16		x
2	INT16		y
2n	STRING16		string
p			unused, p=pad(2n)

CreateColormap

1	78		request opcode
1			alloc
	0	None	
	1	All	
2	4		request length
4	COLORMAP		mid
4	WINDOW		window
4	VISUALID		visual

FreeColormap

1	79	request opcode
1		unused
2	2	request length
4	COLORMAP	cmap

CopyColormapAndFree

1	80	request opcode
1		unused
2	3	request length
4	COLORMAP	mid
4	COLORMAP	src-cmap

InstallColormap

1	81	request opcode
1		unused
2	2	request length
4	COLORMAP	cmap

UninstallColormap

1	82	request opcode
1		unused
2	2	request length
4	COLORMAP	cmap

ListInstalledColormaps

1	83	request opcode
1		unused
2	2	request length
4	WINDOW	window

→

1	1	Reply
1		unused
2	CARD16	sequence number
4	n	reply length
2	n	number of COLORMAPs in cmaps
22		unused
4n	LISTofCOLORMAP	cmaps

AllocColor

1	84	request opcode
1		unused
2	4	request length
4	COLORMAP	cmap
2	CARD16	red
2	CARD16	green
2	CARD16	blue
2		unused

→

1	1	Reply
1		unused
2	CARD16	sequence number
4	0	reply length
2	CARD16	red
2	CARD16	green
2	CARD16	blue
2		unused
4	CARD32	pixel
12		unused

AllocNamedColor

1	85	request opcode
1		unused
2	3+(n+p)/4	request length
4	COLORMAP	cmap
2	n	length of name
2		unused
n	STRING8	name
p		unused, p=pad(n)

→

1	1	Reply
1		unused
2	CARD16	sequence number
4	0	reply length
4	CARD32	pixel
2	CARD16	exact-red
2	CARD16	exact-green
2	CARD16	exact-blue
2	CARD16	visual-red
2	CARD16	visual-green
2	CARD16	visual-blue
8		unused

AllocColorCells

1	86	request opcode
1	BOOL	contiguous
2	3	request length
4	COLORMAP	cmap
2	CARD16	colors
2	CARD16	planes

→

1	1	Reply
1		unused
2	CARD16	sequence number
4	n+m	reply length
2	n	number of CARD32s in pixels
2	m	number of CARD32s in masks
20		unused
4n	LISTofCARD32	pixels
4m	LISTofCARD32	masks

AllocColorPlanes

1	87	request opcode
1	BOOL	contiguous
2	4	request length
4	COLORMAP	cmap
2	CARD16	colors
2	CARD16	reds
2	CARD16	greens
2	CARD16	blues

→

1	1	Reply
1		unused
2	CARD16	sequence number
4	n	reply length
2	n	number of CARD32s in pixels
2		unused
4	CARD32	red-mask
4	CARD32	green-mask
4	CARD32	blue-mask
8		unused
4n	LISTofCARD32	pixels

FreeColors

1	88	request opcode
1		unused
2	3+n	request length
4	COLORMAP	cmap
4	CARD32	plane-mask
4n	LISTofCARD32	pixels

StoreColors

1	89	request opcode
1		unused
2	2+3n	request length
4	COLORMAP	cmap
12n	LISTofCOLORITEM	items

COLORITEM

4	CARD32		pixel
2	CARD16		red
2	CARD16		green
2	CARD16		blue
1			do-red, do-green, do-blue
	#0x01	do-red (1=True, 0=False)	
	#0x02	do-green (1=True, 0=False)	
	#0x04	do-blue (1=True, 0=False)	
	#0xF8	unused	
1			unused

StoreNamedColor

1	90		request opcode
1			do-red, do-green, do-blue
	#0x01	do-red (1=True, 0=False)	
	#0x02	do-green (1=True, 0=False)	
	#0x04	do-blue (1=True, 0=False)	
	#0xF8	unused	
2	4+(n+p)/4		request length
4	COLORMAP		cmap
4	CARD32		pixel
2	n		length of name
2			unused
n	STRING8		name
p			unused, p=pad(n)

QueryColors

1	91	request opcode
1		unused
2	2+n	request length
4	COLORMAP	cmap
4n	LISTofCARD32	pixels

→

1	1	Reply
1		unused
2	CARD16	sequence number
4	2n	reply length
2	n	number of RGBs in colors
22		unused
8n	LISTofRGB	colors

RGB

2	CARD16		red
2	CARD16		green
2	CARD16		blue
2			unused

LookupColor

1	92		request opcode
1			unused
2	3+(n+p)/4		request length
4	COLORMAP		cmap
2	n		length of name
2			unused
n	STRING8		name
p			unused, p=pad(n)

→

1	1		Reply
1			unused
2	CARD16		sequence number
4	0		reply length
2	CARD16		exact-red
2	CARD16		exact-green
2	CARD16		exact-blue
2	CARD16		visual-red
2	CARD16		visual-green
2	CARD16		visual-blue
12			unused

CreateCursor

1	93		request opcode
1			unused
2	8		request length
4	CURSOR		cid
4	PIXMAP		source
4	PIXMAP		mask
	0	None	
2	CARD16		fore-red
2	CARD16		fore-green
2	CARD16		fore-blue
2	CARD16		back-red
2	CARD16		back-green
2	CARD16		back-blue
2	CARD16		x
2	CARD16		y

CreateGlyphCursor

1	94		request opcode
1			unused
2	8		request length
4	CURSOR		cid
4	FONT		source-font
4	FONT		mask-font
	0	None	
2	CARD16		source-char
2	CARD16		mask-char
2	CARD16		fore-red
2	CARD16		fore-green
2	CARD16		fore-blue
2	CARD16		back-red
2	CARD16		back-green
2	CARD16		back-blue

FreeCursor

1	95		request opcode
1			unused
2	2		request length
4	CURSOR		cursor

RecolorCursor

1	96		request opcode
1			unused
2	5		request length
4	CURSOR		cursor
2	CARD16		fore-red
2	CARD16		fore-green
2	CARD16		fore-blue
2	CARD16		back-red
2	CARD16		back-green
2	CARD16		back-blue

QueryBestSize

1	97		request opcode
1			class
	0	Cursor	
	1	Tile	
	2	Stipple	
2	3		request length
4	DRAWABLE		drawable
2	CARD16		width
2	CARD16		height

→

1	1	Reply
1		unused
2	CARD16	sequence number
4	0	reply length
2	CARD16	width
2	CARD16	height
20		unused

QueryExtension

1	98	request opcode
1		unused
2	2+(n+p)/4	request length
2	n	length of name
2		unused
n	STRING8	name
p		unused, p=pad(n)

→

1	1	Reply
1		unused
2	CARD16	sequence number
4	0	reply length
1	BOOL	present
1	CARD8	major-opcode
1	CARD8	first-event
1	CARD8	first-error
20		unused

ListExtensions

1	99	request opcode
1		unused
2	1	request length

→

1	1	Reply
1	CARD8	number of STRs in names
2	CARD16	sequence number
4	(n+p)/4	reply length
24		unused
n	LISTofSTR	names
p		unused, p=pad(n)

ChangeKeyboardMapping

1	100	request opcode
1	n	keycode-count
2	2+nm	request length
1	KEYCODE	first-keycode
1	m	keysyms-per-keycode
2		unused
4nm	LISTofKEYSYM	keysyms

GetKeyboardMapping

1	101	request opcode
1		unused
2	2	request length
1	KEYCODE	first-keycode
1	m	count
2		unused

→

1	1	Reply
1	n	keysyms-per-keycode
2	CARD16	sequence number
4	nm	reply length (m = count field from the request)
24		unused
4nm	LISTofKEYSYM	keysyms

ChangeKeyboardControl

1	102		request opcode
1			unused
2	2+n		request length
4	BITMASK		value-mask (has n bits set to 1)
	#0x0001	key-click-percent	
	#0x0002	bell-percent	
	#0x0004	bell-pitch	
	#0x0008	bell-duration	
	#0x0010	led	
	#0x0020	led-mode	
	#0x0040	key	
	#0x0080	auto-repeat-mode	
4n	LISTofVALUE		value-list

VALUE

1	INT8		key-click-percent
1	INT8		bell-percent
2	INT16		bell-pitch
2	INT16		bell-duration
1	CARD8		led
1			led-mode
	0	Off	
	1	On	
1	KEYCODE		key
1			auto-repeat-mode
	0	Off	
	1	On	
	2	Default	

GetKeyboardControl

1	103		request opcode
1			unused
2	1		request length

→

1	1		Reply
1			global-auto-repeat
	0	Off	
	1	On	
2	CARD16		sequence number
4	5		reply length
4	CARD32		led-mask
1	CARD8		key-click-percent
1	CARD8		bell-percent
2	CARD16		bell-pitch
2	CARD16		bell-duration
2			unused
32	LISTofCARD8		auto-repeats

Bell

1	104	request opcode
1	INT8	percent
2	1	request length

ChangePointerControl

1	105		request opcode
1			unused
2	3		request length
2	INT16		acceleration-numerator
2	INT16		acceleration-denominator
2	INT16		threshold
1	BOOL		do-acceleration
1	BOOL		do-threshold

GetPointerControl

1	106		request opcode
1			unused
2	1		request length

→

1	1		Reply
1			unused
2	CARD16		sequence number
4	0		reply length
2	CARD16		acceleration-numerator
2	CARD16		acceleration-denominator
2	CARD16		threshold
18			unused

SetScreenSaver

1	107		request opcode
1			unused
2	3		request length
2	INT16		timeout
2	INT16		interval
1			prefer-blanking
	0	No	
	1	Yes	
	2	Default	
1			allow-exposures
	0	No	
	1	Yes	
	2	Default	
2			unused

GetScreenSaver

1	108		request opcode
1			unused
2	1		request length

→

1	1		Reply
1			unused
2	CARD16		sequence number
4	0		reply length
2	CARD16		timeout
2	CARD16		interval
1			prefer-blanking
	0	No	
	1	Yes	
1			allow-exposures
	0	No	
	1	Yes	
18			unused

ChangeHosts

1	109		request opcode
1			mode
	0	Insert	
	1	Delete	
2	2+(n+p)/4		request length
1			family
	0	Internet	
	1	DECnet	
	2	Chaos	
1			unused
2	n		length of address
n	LISTofCARD8		address
p			unused, p=pad(n)

ListHosts

1	110		request opcode
1			unused
2	1		request length

→

1	1		Reply
1			mode
	0	Disabled	
	1	Enabled	
2	CARD16		sequence number
4	n/4		reply length
2	CARD16		number of HOSTs in hosts
22			unused
n	LISTofHOST		hosts (n always a multiple of 4)

SetAccessControl

1	111		request opcode
1			mode
	0	Disable	
	1	Enable	
2	1		request length

SetCloseDownMode

1	112		request opcode
1			mode
	0	Destroy	
	1	RetainPermanent	
	2	RetainTemporary	
2	1		request length

KillClient

1	113		request opcode
1			unused
2	2		request length
4	CARD32		resource
	0	AllTemporary	

RotateProperties

1	114		request opcode
1			unused
2	3+n		request length
4	WINDOW		window
2	n		number of properties
2	INT16		delta
4n	LISTofATOM		properties

ForceScreenSaver

1	115		request opcode
1			mode
	0	Reset	
	1	Activate	
2	1		request length

SetPointerMapping

1	116		request opcode
1	n		length of map
2	1+(n+p)/4		request length
n	LISTofCARD8		map
p			unused, p=pad(n)

→

1	1		Reply
1			status
	0	Success	
	1	Busy	
2	CARD16		sequence number
4	0		reply length
24			unused

GetPointerMapping

1	117		request opcode
1			unused
2	1		request length

→

1	1		Reply
1	n		length of map
2	CARD16		sequence number
4	(n+p)/4		reply length
24			unused
n	LISTofCARD8		map
p			unused, p=pad(n)

SetModifierMapping

1	118		request opcode
1	n		keycodes-per-modifier
2	1+2n		request length
8n	LISTofKEYCODE		keycodes

→

1	1		Reply
1			status
	0	Success	
	1	Busy	
	2	Failed	
2	CARD16		sequence number
4	0		reply length
24			unused

GetModifierMapping

1	119		request opcode
1			unused
2	1		request length

→

1	1		Reply
1	n		keycodes-per-modifier
2	CARD16		sequence number
4	2n		reply length
24			unused
8n	LISTofKEYCODE		keycodes

NoOperation

1	127		request opcode
1			unused
2	1+n		request length
4n			unused

5.6 Events

This section presents each event in the core X protocol, in order of the event opcode.

KeyPress

1	2		code
1	KEYCODE		detail
2	CARD16		sequence number
4	TIMESTAMP		time
4	WINDOW		root
4	WINDOW		event
4	WINDOW		child
	0	None	
2	INT16		root-x
2	INT16		root-y
2	INT16		event-x
2	INT16		event-y
2	SETofKEYBUTMASK		state
1	BOOL		same-screen
1			unused

KeyRelease

1	3		code
1	KEYCODE		detail
2	CARD16		sequence number
4	TIMESTAMP		time
4	WINDOW		root
4	WINDOW		event
4	WINDOW		child
	0	None	
2	INT16		root-x
2	INT16		root-y
2	INT16		event-x
2	INT16		event-y
2	SETofKEYBUTMASK		state
1	BOOL		same-screen
1			unused

ButtonPress

1	4		code
1	BUTTON		detail
2	CARD16		sequence number
4	TIMESTAMP		time
4	WINDOW		root
4	WINDOW		event
4	WINDOW		child
	0	None	
2	INT16		root-x
2	INT16		root-y
2	INT16		event-x
2	INT16		event-y
2	SETofKEYBUTMASK		state
1	BOOL		same-screen
1			unused

ButtonRelease

1	5		code
1	BUTTON		detail
2	CARD16		sequence number
4	TIMESTAMP		time
4	WINDOW		root
4	WINDOW		event
4	WINDOW		child
	0	None	
2	INT16		root-x
2	INT16		root-y
2	INT16		event-x
2	INT16		event-y
2	SETofKEYBUTMASK		state
1	BOOL		same-screen
1			unused

MotionNotify

1	6		code
1			detail
	0	Normal	
	1	Hint	
2	CARD16		sequence number
4	TIMESTAMP		time
4	WINDOW		root
4	WINDOW		event
4	WINDOW		child
	0	None	
2	INT16		root-x
2	INT16		root-y
2	INT16		event-x
2	INT16		event-y
2	SETofKEYBUTMASK		state
1	BOOL		same-screen
1			unused

EnterNotify

1	7		code
1			detail
	0	Ancestor	
	1	Virtual	
	2	Inferior	
	3	Nonlinear	
	4	NonlinearVirtual	
2	CARD16		sequence number
4	TIMESTAMP		time
4	WINDOW		root
4	WINDOW		event
4	WINDOW		child
	0	None	
2	INT16		root-x
2	INT16		root-y
2	INT16		event-x
2	INT16		event-y
2	SETofKEYBUTMASK		state
1			mode
	0	Normal	
	1	Grab	
	2	Ungrab	
1			same-screen, focus
	#0x01	focus (1=True, 0=False)	
	#0x02	same-screen (1=True, 0=False)	
	#0xFC	unused	

LeaveNotify

1	8		code
1			detail
	0	Ancestor	
	1	Virtual	
	2	Inferior	
	3	Nonlinear	
	4	NonlinearVirtual	
2	CARD16		sequence number
4	TIMESTAMP		time
4	WINDOW		root
4	WINDOW		event
4	WINDOW		child
	0	None	
2	INT16		root-x
2	INT16		root-y
2	INT16		event-x
2	INT16		event-y
2	SETofKEYBUTMASK		state
1			mode
	0	Normal	
	1	Grab	
	2	Ungrab	
1			same-screen, focus
	#0x01	focus (1=True, 0=False)	
	#0x02	same-screen (1=True, 0=False)	
	#0xFC	unused	

FocusIn

1	9		code
1			detail
	0	Ancestor	
	1	Virtual	
	2	Inferior	
	3	Nonlinear	
	4	NonlinearVirtual	
	5	Pointer	
	6	PointerRoot	
	7	None	
2	CARD16		sequence number
4	WINDOW		event
1			mode
	0	Normal	
	1	Grab	
	2	Ungrab	
	3	WhileGrabbed	
23			unused

FocusOut

1	10		code
1			detail
	0	Ancestor	
	1	Virtual	
	2	Inferior	
	3	Nonlinear	
	4	NonlinearVirtual	
	5	Pointer	
	6	PointerRoot	
	7	None	
2	CARD16		sequence number
4	WINDOW		event
1			mode
	0	Normal	
	1	Grab	
	2	Ungrab	
	3	WhileGrabbed	
23			unused

KeymapNotify

1	11	code
31	LISTofCARD8	keys (byte for keycodes 0-7 is omitted)

Expose

1	12	code
1		unused
2	CARD16	sequence number
4	WINDOW	window
2	CARD16	x
2	CARD16	y
2	CARD16	width
2	CARD16	height
2	CARD16	count
14		unused

GraphicsExpose

1	13		code
1			unused
2	CARD16		sequence number
4	DRAWABLE		drawable
2	CARD16		x
2	CARD16		y
2	CARD16		width
2	CARD16		height
2	CARD16		minor-opcode
2	CARD16		count
1	CARD8		major-opcode
11			unused

NoExpose

1	14		code
1			unused
2	CARD16		sequence number
4	DRAWABLE		drawable
2	CARD16		minor-opcode
1	CARD8		major-opcode
21			unused

VisibilityNotify

1	15		code
1			unused
2	CARD16		sequence number
4	WINDOW		window
1			state
	0	Unobscured	
	1	PartiallyObscured	
	2	FullyObscured	
23			unused

CreateNotify

1	16		code
1			unused
2	CARD16		sequence number
4	WINDOW		parent
4	WINDOW		window
2	INT16		x
2	INT16		y
2	CARD16		width
2	CARD16		height
2	CARD16		border-width
1	BOOL		override-redirect
9			unused

DestroyNotify

1	17	code
1		unused
2	CARD16	sequence number
4	WINDOW	event
4	WINDOW	window
20		unused

UnmapNotify

1	18	code
1		unused
2	CARD16	sequence number
4	WINDOW	event
4	WINDOW	window
1	BOOL	from-configure
19		unused

MapNotify

1	19	code
1		unused
2	CARD16	sequence number
4	WINDOW	event
4	WINDOW	window
1	BOOL	override-redirect
19		unused

MapRequest

1	20	code
1		unused
2	CARD16	sequence number
4	WINDOW	parent
4	WINDOW	window
20		unused

ReparentNotify

1	21	code
1		unused
2	CARD16	sequence number
4	WINDOW	event
4	WINDOW	window
4	WINDOW	parent
2	INT16	x
2	INT16	y
1	BOOL	override-redirect
11		unused

ConfigureNotify

1	22		code
1			unused
2	CARD16		sequence number
4	WINDOW		event
4	WINDOW		window
4	WINDOW		above-sibling
	0	None	
2	INT16		x
2	INT16		y
2	CARD16		width
2	CARD16		height
2	CARD16		border-width
1	BOOL		override-redirect
5			unused

ConfigureRequest

1	23		code
1			stack-mode
	0	Above	
	1	Below	
	2	TopIf	
	3	BottomIf	
	4	Opposite	
2	CARD16		sequence number
4	WINDOW		parent
4	WINDOW		window
4	WINDOW		sibling
	0	None	
2	INT16		x
2	INT16		y
2	CARD16		width
2	CARD16		height
2	CARD16		border-width
2	BITMASK		value-mask
	#0x0001	x	
	#0x0002	y	
	#0x0004	width	
	#0x0008	height	
	#0x0010	border-width	
	#0x0020	sibling	
	#0x0040	stack-mode	
4			unused

GravityNotify

1	24		code
1			unused
2	CARD16		sequence number
4	WINDOW		event
4	WINDOW		window
2	INT16		x
2	INT16		y
16			unused

ResizeRequest

1	25		code
1			unused
2	CARD16		sequence number
4	WINDOW		window
2	CARD16		width
2	CARD16		height
20			unused

CirculateNotify

1	26		code
1			unused
2	CARD16		sequence number
4	WINDOW		event
4	WINDOW		window
4			unused
1			place
	0	Top	
	1	Bottom	
15			unused

CirculateRequest

1	27		code
1			unused
2	CARD16		sequence number
4	WINDOW		parent
4	WINDOW		window
4			unused
1			place
	0	Top	
	1	Bottom	
15			unused

PropertyNotify

1	28		code
1			unused
2	CARD16		sequence number
4	WINDOW		window
4	ATOM		atom
4	TIMESTAMP		time
1			state
	0	NewValue	
	1	Deleted	
15			unused

SelectionClear

1	29		code
1			unused
2	CARD16		sequence number
4	TIMESTAMP		time
4	WINDOW		owner
4	ATOM		selection
16			unused

SelectionRequest

1	30		code
1			unused
2	CARD16		sequence number
4	TIMESTAMP		time
	0	CurrentTime	
4	WINDOW		owner
4	WINDOW		requestor
4	ATOM		selection
4	ATOM		target
4	ATOM		property
	0	None	
4			unused

SelectionNotify

1	31		code
1			unused
2	CARD16		sequence number
4	TIMESTAMP		time
	0	CurrentTime	
4	WINDOW		requestor
4	ATOM		selection
4	ATOM		target
4	ATOM		property
	0	None	
8			unused

ColormapNotify

1	32		code
1			unused
2	CARD16		sequence number
4	WINDOW		window
4	COLORMAP		colormap
	0	None	
1	BOOL		new
1			state
	0	Uninstalled	
	1	Installed	
18			unused

ClientMessage

1	33		code
1	CARD8		format
2	CARD16		sequence number
4	WINDOW		window
4	ATOM		type
20			data

MappingNotify

1	34		code
1			unused
2	CARD16		sequence number
1			request
	0	Modifier	
	1	Keyboard	
	2	Pointer	
1	KEYCODE		first-keycode
1	CARD8		count
25			unused

5.7 Errors

This section presents each error in the core X protocol, in order of the error code.

[REQUEST]

1	0	Error
1	1	code
2	CARD16	sequence number
4		unused
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

[VALUE]

1	0	Error
1	2	code
2	CARD16	sequence number
4	unspecified	bad value
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

[WINDOW]

1	0	Error
1	3	code
2	CARD16	sequence number
4	CARD32	bad resource ID
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

[PIXMAP]

1	0	Error
1	4	code
2	CARD16	sequence number
4	CARD32	bad resource ID
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

[ATOM]

1	0	Error
1	5	code
2	CARD16	sequence number
4	CARD32	bad atom ID
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

[CURSOR]

1	0	Error
1	6	code
2	CARD16	sequence number
4	CARD32	bad resource ID
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

[FONT]

1	0	Error
1	7	code
2	CARD16	sequence number
4	CARD32	bad resource ID
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

[MATCH]

1	0	Error
1	8	code
2	CARD16	sequence number
4		unused
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

[DRAWABLE]

1	0	Error
1	9	code
2	CARD16	sequence number
4	CARD32	bad resource ID
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

[ACCESS]

1	0	Error
1	10	code
2	CARD16	sequence number
4		unused
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

[ALLOC]

1	0	Error
1	11	code
2	CARD16	sequence number
4		unused
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

[COLORMAP]

1	0	Error
1	12	code
2	CARD16	sequence number
4	CARD32	bad resource ID
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

[GCONTEXT]

1	0	Error
1	13	code
2	CARD16	sequence number
4	CARD32	bad resource ID
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

[IDCHOICE]

1	0	Error
1	14	code
2	CARD16	sequence number
4	CARD32	bad resource ID
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

[NAME]

1	0	Error
1	15	code
2	CARD16	sequence number
4		unused
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

[LENGTH]

1	0	Error
1	16	code
2	CARD16	sequence number
4		unused
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

[IMPLEMENTATION]

1	0	Error
1	17	code
2	CARD16	sequence number
4		unused
2	CARD16	minor opcode
1	CARD8	major opcode
21		unused

Appendix A

KEYSYM Encoding

For convenience, KEYSYM values can be viewed as split into 4 bytes:

- Byte 1 (for the purposes of this encoding) is the most-significant 5 bits (because of the 29-bit effective values).
- Byte 2 is the next most-significant 8 bits.
- Byte 3 is the next most-significant 8 bits.
- Byte 4 is the least-significant 8 bits.

KEYSYMs with the most-significant bit set (of the 29 bits) are reserved as implementation-dependent.

There are two special KEYSYM values: *NoSymbol* and *VoidSymbol*. They are used to indicate the absence of symbols.

Byte 1	Byte 2	Byte 3	Byte 4	Name
0	0	0	0	*NoSymbol*
0	255	255	255	*VoidSymbol*

All other standard KEYSYM values have bytes 1 and 2 set to 0. Byte 3 indicates a character code set and byte 4 selects a character within that set.

Byte 3	Character Set
0	ISO Latin-1
1	ISO Latin-2
2	ISO Latin-3
3	ISO Latin-4
4	Kana
5	Arabic
6	Cyrillic
7	Greek
8	Technical
9	Special
10	Publishing
11	APL
12	Hebrew
13	Thai
14	Korean
15	ISO Latin-5*
16	ISO Latin-6*
17	ISO Latin-7*
18	ISO Latin-8*
19	ISO Latin-9*
32	Currency

Byte 3	Character Set
255	Keyboard

Character sets may contain gaps where codes have been removed that were duplicates with codes in previous character sets (that is, character sets for which byte 3 has a lower value). The ordering between the sets (byte 3) has no meaning beyond dealing with duplicate coding.

As far as possible, KEYSYM values (byte 4) follow the character set encoding standards, except for the Greek and Cyrillic KEYSYMs which are based on early draft standards. In the Latin-1 to Latin-4 sets, all duplicate glyphs occupy the same code position. However, duplicates between Greek and Technical do not occupy the same code position.[9]

The 94 and 96-character code sets occupy the right-hand quadrant (decimal 129 through 256), so the ASCII subset has a unique encoding across byte 4, which corresponds to the ASCII character code. However, this does not apply to all values in the Keyboard set (byte 3 = 255).

The Keyboard set (byte 3 = 255) is a collection of commonly occurring keys on keyboards. Within this set, the keypad symbols are generally duplicates of symbols found on keys on the main part of the keyboard, but they are distinguished here because they often have distinguishable semantics associated with them.

This specification captures keys from a variety of keyboards, folding likely aliases into the same KEYSYM (for example, merging Del, DEL, and Delete into a single KEYSYM), rather than creating a KEYSYM for every unique key.[10]

In the tables below, **Code Position** is a restatement of byte 4 of the KEYSYM value as most-significant / least-significant 4-bit values.

In all cases, the KEYSYM value is:

byte3 * 256 + *byte4*

---

* This appendix does not define any new keysyms for these character sets. All keysyms in these character sets are reserved.

9. **Application Usage:** Applications that use the Latin-2, Latin-3, Latin-4, Greek, Cyrillic, or Technical sets may find it convenient to use arrays to transform the KEYSYMs.

10. **Application Usage:** For example, providing a single, keyboard-independent KEYSYM for Delete lets an application implement a deletion function and expect reasonable bindings on a wide set of workstations.

Byte 3	Byte 4	Code Position	Name	Set
000	032	02/00	SPACE	Latin-1
000	033	02/01	EXCLAMATION POINT	Latin-1
000	034	02/02	QUOTATION MARK	Latin-1
000	035	02/03	NUMBER SIGN	Latin-1
000	036	02/04	DOLLAR SIGN	Latin-1
000	037	02/05	PERCENT SIGN	Latin-1
000	038	02/06	AMPERSAND	Latin-1
000	039	02/07	APOSTROPHE	Latin-1
000	040	02/08	LEFT PARENTHESIS	Latin-1
000	041	02/09	RIGHT PARENTHESIS	Latin-1
000	042	02/10	ASTERISK	Latin-1
000	043	02/11	PLUS SIGN	Latin-1
000	044	02/12	COMMA	Latin-1
000	045	02/13	MINUS SIGN	Latin-1
000	046	02/14	FULL STOP	Latin-1
000	047	02/15	SOLIDUS	Latin-1
000	048	03/00	DIGIT ZERO	Latin-1
000	049	03/01	DIGIT ONE	Latin-1
000	050	03/02	DIGIT TWO	Latin-1
000	051	03/03	DIGIT THREE	Latin-1
000	052	03/04	DIGIT FOUR	Latin-1
000	053	03/05	DIGIT FIVE	Latin-1
000	054	03/06	DIGIT SIX	Latin-1
000	055	03/07	DIGIT SEVEN	Latin-1
000	056	03/08	DIGIT EIGHT	Latin-1
000	057	03/09	DIGIT NINE	Latin-1
000	058	03/10	COLON	Latin-1
000	059	03/11	SEMICOLON	Latin-1
000	060	03/12	LESS THAN SIGN	Latin-1
000	061	03/13	EQUALS SIGN	Latin-1
000	062	03/14	GREATER THAN SIGN	Latin-1
000	063	03/15	QUESTION MARK	Latin-1
000	064	04/00	COMMERCIAL AT	Latin-1
000	065	04/01	LATIN CAPITAL LETTER A	Latin-1
000	066	04/02	LATIN CAPITAL LETTER B	Latin-1
000	067	04/03	LATIN CAPITAL LETTER C	Latin-1
000	068	04/04	LATIN CAPITAL LETTER D	Latin-1
000	069	04/05	LATIN CAPITAL LETTER E	Latin-1
000	070	04/06	LATIN CAPITAL LETTER F	Latin-1
000	071	04/07	LATIN CAPITAL LETTER G	Latin-1
000	072	04/08	LATIN CAPITAL LETTER H	Latin-1
000	073	04/09	LATIN CAPITAL LETTER I	Latin-1
000	074	04/10	LATIN CAPITAL LETTER J	Latin-1
000	075	04/11	LATIN CAPITAL LETTER K	Latin-1
000	076	04/12	LATIN CAPITAL LETTER L	Latin-1
000	077	04/13	LATIN CAPITAL LETTER M	Latin-1
000	078	04/14	LATIN CAPITAL LETTER N	Latin-1
000	079	04/15	LATIN CAPITAL LETTER O	Latin-1
000	080	05/00	LATIN CAPITAL LETTER P	Latin-1
000	081	05/01	LATIN CAPITAL LETTER Q	Latin-1
000	082	05/02	LATIN CAPITAL LETTER R	Latin-1
000	083	05/03	LATIN CAPITAL LETTER S	Latin-1

Byte 3	Byte 4	Code Position	Name	Set
000	084	05/04	LATIN CAPITAL LETTER T	Latin-1
000	085	05/05	LATIN CAPITAL LETTER U	Latin-1
000	086	05/06	LATIN CAPITAL LETTER V	Latin-1
000	087	05/07	LATIN CAPITAL LETTER W	Latin-1
000	088	05/08	LATIN CAPITAL LETTER X	Latin-1
000	089	05/09	LATIN CAPITAL LETTER Y	Latin-1
000	090	05/10	LATIN CAPITAL LETTER Z	Latin-1
000	091	05/11	LEFT SQUARE BRACKET	Latin-1
000	092	05/12	REVERSE SOLIDUS	Latin-1
000	093	05/13	RIGHT SQUARE BRACKET	Latin-1
000	094	05/14	CIRCUMFLEX ACCENT	Latin-1
000	095	05/15	LOW LINE	Latin-1
000	096	06/00	GRAVE ACCENT	Latin-1
000	097	06/01	LATIN SMALL LETTER a	Latin-1
000	098	06/02	LATIN SMALL LETTER b	Latin-1
000	099	06/03	LATIN SMALL LETTER c	Latin-1
000	100	06/04	LATIN SMALL LETTER d	Latin-1
000	101	06/05	LATIN SMALL LETTER e	Latin-1
000	102	06/06	LATIN SMALL LETTER f	Latin-1
000	103	06/07	LATIN SMALL LETTER g	Latin-1
000	104	06/08	LATIN SMALL LETTER h	Latin-1
000	105	06/09	LATIN SMALL LETTER i	Latin-1
000	106	06/10	LATIN SMALL LETTER j	Latin-1
000	107	06/11	LATIN SMALL LETTER k	Latin-1
000	108	06/12	LATIN SMALL LETTER l	Latin-1
000	109	06/13	LATIN SMALL LETTER m	Latin-1
000	110	06/14	LATIN SMALL LETTER n	Latin-1
000	111	06/15	LATIN SMALL LETTER o	Latin-1
000	112	07/00	LATIN SMALL LETTER p	Latin-1
000	113	07/01	LATIN SMALL LETTER q	Latin-1
000	114	07/02	LATIN SMALL LETTER r	Latin-1
000	115	07/03	LATIN SMALL LETTER s	Latin-1
000	116	07/04	LATIN SMALL LETTER t	Latin-1
000	117	07/05	LATIN SMALL LETTER u	Latin-1
000	118	07/06	LATIN SMALL LETTER v	Latin-1
000	119	07/07	LATIN SMALL LETTER w	Latin-1
000	120	07/08	LATIN SMALL LETTER x	Latin-1
000	121	07/09	LATIN SMALL LETTER y	Latin-1
000	122	07/10	LATIN SMALL LETTER z	Latin-1
000	123	07/11	LEFT CURLY BRACKET	Latin-1
000	124	07/12	VERTICAL LINE	Latin-1
000	125	07/13	RIGHT CURLY BRACKET	Latin-1
000	126	07/14	TILDE	Latin-1
000	160	10/00	NO-BREAK SPACE	Latin-1
000	161	10/01	INVERTED EXCLAMATION MARK	Latin-1
000	162	10/02	CENT SIGN	Latin-1
000	163	10/03	POUND SIGN	Latin-1
000	164	10/04	CURRENCY SIGN	Latin-1
000	165	10/05	YEN SIGN	Latin-1
000	166	10/06	BROKEN VERTICAL BAR	Latin-1
000	167	10/07	SECTION SIGN (Also known as Paragraph sign)	Latin-1
000	168	10/08	DIAERESIS	Latin-1

Byte 3	Byte 4	Code Position	Name	Set
000	169	10/09	COPYRIGHT SIGN	Latin-1
000	170	10/10	FEMININE ORDINAL INDICATOR	Latin-1
000	171	10/11	LEFT ANGLE QUOTATION MARK	Latin-1
000	172	10/12	NOT SIGN	Latin-1
000	173	10/13	HYPHEN	Latin-1
000	174	10/14	REGISTERED TRADEMARK SIGN	Latin-1
000	175	10/15	MACRON	Latin-1
000	176	11/00	DEGREE SIGN, RING ABOVE	Latin-1
000	177	11/01	PLUS-MINUS SIGN	Latin-1
000	178	11/02	SUPERSCRIPT TWO	Latin-1
000	179	11/03	SUPERSCRIPT THREE	Latin-1
000	180	11/04	ACUTE ACCENT	Latin-1
000	181	11/05	MICRO SIGN	Latin-1
000	182	11/06	PARAGRAPH SIGN (Also known as Pilcrow sign)	Latin-1
000	183	11/07	MIDDLE DOT	Latin-1
000	184	11/08	CEDILLA	Latin-1
000	185	11/09	SUPERSCRIPT ONE	Latin-1
000	186	11/10	MASCULINE ORDINAL INDICATOR	Latin-1
000	187	11/11	RIGHT ANGLE QUOTATION MARK	Latin-1
000	188	11/12	VULGAR FRACTION ONE QUARTER	Latin-1
000	189	11/13	VULGAR FRACTION ONE HALF	Latin-1
000	190	11/14	VULGAR FRACTION THREE QUARTERS	Latin-1
000	191	11/15	INVERTED QUESTION MARK	Latin-1
000	192	12/00	LATIN CAPITAL LETTER A WITH GRAVE ACCENT	Latin-1
000	193	12/01	LATIN CAPITAL LETTER A WITH ACUTE ACCENT	Latin-1
000	194	12/02	LATIN CAPITAL LETTER A WITH CIRCUMFLEX ACCENT	Latin-1
000	195	12/03	LATIN CAPITAL LETTER A WITH TILDE	Latin-1
000	196	12/04	LATIN CAPITAL LETTER A WITH DIAERESIS	Latin-1
000	197	12/05	LATIN CAPITAL LETTER A WITH RING ABOVE	Latin-1
000	198	12/06	LATIN CAPITAL DIPHTHONG AE	Latin-1
000	199	12/07	LATIN CAPITAL LETTER C WITH CEDILLA	Latin-1
000	200	12/08	LATIN CAPITAL LETTER E WITH GRAVE ACCENT	Latin-1
000	201	12/09	LATIN CAPITAL LETTER E WITH ACUTE ACCENT	Latin-1
000	202	12/10	LATIN CAPITAL LETTER E WITH CIRCUMFLEX ACCENT	Latin-1
000	203	12/11	LATIN CAPITAL LETTER E WITH DIAERESIS	Latin-1
000	204	12/12	LATIN CAPITAL LETTER I WITH GRAVE ACCENT	Latin-1
000	205	12/13	LATIN CAPITAL LETTER I WITH ACUTE ACCENT	Latin-1
000	206	12/14	LATIN CAPITAL LETTER I WITH CIRCUMFLEX ACCENT	Latin-1
000	207	12/15	LATIN CAPITAL LETTER I WITH DIAERESIS	Latin-1
000	208	13/00	ICELANDIC CAPITAL LETTER ETH	Latin-1
000	209	13/01	LATIN CAPITAL LETTER N WITH TILDE	Latin-1
000	210	13/02	LATIN CAPITAL LETTER O WITH GRAVE ACCENT	Latin-1
000	211	13/03	LATIN CAPITAL LETTER O WITH ACUTE ACCENT	Latin-1
000	212	13/04	LATIN CAPITAL LETTER O WITH CIRCUMFLEX ACCENT	Latin-1
000	213	13/05	LATIN CAPITAL LETTER O WITH TILDE	Latin-1
000	214	13/06	LATIN CAPITAL LETTER O WITH DIAERESIS	Latin-1
000	215	13/07	MULTIPLICATION SIGN	Latin-1
000	216	13/08	LATIN CAPITAL LETTER O WITH OBLIQUE STROKE	Latin-1
000	217	13/09	LATIN CAPITAL LETTER U WITH GRAVE ACCENT	Latin-1
000	218	13/10	LATIN CAPITAL LETTER U WITH ACUTE ACCENT	Latin-1
000	219	13/11	LATIN CAPITAL LETTER U WITH CIRCUMFLEX ACCENT	Latin-1
000	220	13/12	LATIN CAPITAL LETTER U WITH DIAERESIS	Latin-1

Byte 3	Byte 4	Code Position	Name	Set
000	221	13/13	LATIN CAPITAL LETTER Y WITH ACUTE ACCENT	Latin-1
000	222	13/14	ICELANDIC CAPITAL LETTER THORN	Latin-1
000	223	13/15	GERMAN SMALL LETTER SHARP s	Latin-1
000	224	14/00	LATIN SMALL LETTER a WITH GRAVE ACCENT	Latin-1
000	225	14/01	LATIN SMALL LETTER a WITH ACUTE ACCENT	Latin-1
000	226	14/02	LATIN SMALL LETTER a WITH CIRCUMFLEX ACCENT	Latin-1
000	227	14/03	LATIN SMALL LETTER a WITH TILDE	Latin-1
000	228	14/04	LATIN SMALL LETTER a WITH DIAERESIS	Latin-1
000	229	14/05	LATIN SMALL LETTER a WITH RING ABOVE	Latin-1
000	230	14/06	LATIN SMALL DIPHTHONG ae	Latin-1
000	231	14/07	LATIN SMALL LETTER c WITH CEDILLA	Latin-1
000	232	14/08	LATIN SMALL LETTER e WITH GRAVE ACCENT	Latin-1
000	233	14/09	LATIN SMALL LETTER e WITH ACUTE ACCENT	Latin-1
000	234	14/10	LATIN SMALL LETTER e WITH CIRCUMFLEX ACCENT	Latin-1
000	235	14/11	LATIN SMALL LETTER e WITH DIAERESIS	Latin-1
000	236	14/12	LATIN SMALL LETTER i WITH GRAVE ACCENT	Latin-1
000	237	14/13	LATIN SMALL LETTER i WITH ACUTE ACCENT	Latin-1
000	238	14/14	LATIN SMALL LETTER i WITH CIRCUMFLEX ACCENT	Latin-1
000	239	14/15	LATIN SMALL LETTER i WITH DIAERESIS	Latin-1
000	240	15/00	ICELANDIC SMALL LETTER ETH	Latin-1
000	241	15/01	LATIN SMALL LETTER n WITH TILDE	Latin-1
000	242	15/02	LATIN SMALL LETTER o WITH GRAVE ACCENT	Latin-1
000	243	15/03	LATIN SMALL LETTER o WITH ACUTE ACCENT	Latin-1
000	244	15/04	LATIN SMALL LETTER o WITH CIRCUMFLEX ACCENT	Latin-1
000	245	15/05	LATIN SMALL LETTER o WITH TILDE	Latin-1
000	246	15/06	LATIN SMALL LETTER o WITH DIAERESIS	Latin-1
000	247	15/07	DIVISION SIGN	Latin-1
000	248	15/08	LATIN SMALL LETTER o WITH OBLIQUE STROKE	Latin-1
000	249	15/09	LATIN SMALL LETTER u WITH GRAVE ACCENT	Latin-1
000	250	15/10	LATIN SMALL LETTER u WITH ACUTE ACCENT	Latin-1
000	251	15/11	LATIN SMALL LETTER u WITH CIRCUMFLEX ACCENT	Latin-1
000	252	15/12	LATIN SMALL LETTER u WITH DIAERESIS	Latin-1
000	253	15/13	LATIN SMALL LETTER y WITH ACUTE ACCENT	Latin-1
000	254	15/14	ICELANDIC SMALL LETTER THORN	Latin-1
000	255	15/15	LATIN SMALL LETTER y WITH DIAERESIS	Latin-1
001	161	10/01	LATIN CAPITAL LETTER A WITH OGONEK	Latin-2
001	162	10/02	BREVE	Latin-2
001	163	10/03	LATIN CAPITAL LETTER L WITH STROKE	Latin-2
001	165	10/05	LATIN CAPITAL LETTER L WITH CARON	Latin-2
001	166	10/06	LATIN CAPITAL LETTER S WITH ACUTE ACCENT	Latin-2
001	169	10/09	LATIN CAPITAL LETTER S WITH CARON	Latin-2
001	170	10/10	LATIN CAPITAL LETTER S WITH CEDILLA	Latin-2
001	171	10/11	LATIN CAPITAL LETTER T WITH CARON	Latin-2
001	172	10/12	LATIN CAPITAL LETTER Z WITH ACUTE ACCENT	Latin-2
001	174	10/14	LATIN CAPITAL LETTER Z WITH CARON	Latin-2
001	175	10/15	LATIN CAPITAL LETTER Z WITH DOT ABOVE	Latin-2
001	177	11/01	LATIN SMALL LETTER a WITH OGONEK	Latin-2
001	178	11/02	OGONEK	Latin-2
001	179	11/03	LATIN SMALL LETTER l WITH STROKE	Latin-2
001	181	11/05	LATIN SMALL LETTER l WITH CARON	Latin-2
001	182	11/06	LATIN SMALL LETTER s WITH ACUTE ACCENT	Latin-2
001	183	11/07	CARON	Latin-2

Byte 3	Byte 4	Code Position	Name	Set
001	185	11/09	LATIN SMALL LETTER s WITH CARON	Latin-2
001	186	11/10	LATIN SMALL LETTER s WITH CEDILLA	Latin-2
001	187	11/11	LATIN SMALL LETTER t WITH CARON	Latin-2
001	188	11/12	LATIN SMALL LETTER z WITH ACUTE ACCENT	Latin-2
001	189	11/13	DOUBLE ACUTE ACCENT	Latin-2
001	190	11/14	LATIN SMALL LETTER z WITH CARON	Latin-2
001	191	11/15	LATIN SMALL LETTER z WITH DOT ABOVE	Latin-2
001	192	12/00	LATIN CAPITAL LETTER R WITH ACUTE ACCENT	Latin-2
001	195	12/03	LATIN CAPITAL LETTER A WITH BREVE	Latin-2
001	197	12/05	LATIN CAPITAL LETTER L WITH ACUTE ACCENT	Latin-2
001	198	12/06	LATIN CAPITAL LETTER C WITH ACUTE ACCENT	Latin-2
001	200	12/08	LATIN CAPITAL LETTER C WITH CARON	Latin-2
001	202	12/10	LATIN CAPITAL LETTER E WITH OGONEK	Latin-2
001	204	12/12	LATIN CAPITAL LETTER E WITH CARON	Latin-2
001	207	12/15	LATIN CAPITAL LETTER D WITH CARON	Latin-2
001	208	13/00	LATIN CAPITAL LETTER D WITH STROKE	Latin-2
001	209	13/01	LATIN CAPITAL LETTER N WITH ACUTE ACCENT	Latin-2
001	210	13/02	LATIN CAPITAL LETTER N WITH CARON	Latin-2
001	213	13/05	LATIN CAPITAL LETTER O WITH DOUBLE ACUTE ACCENT	Latin-2
001	216	13/08	LATIN CAPITAL LETTER R WITH CARON	Latin-2
001	217	13/09	LATIN CAPITAL LETTER U WITH RING ABOVE	Latin-2
001	219	13/11	LATIN CAPITAL LETTER U WITH DOUBLE ACUTE ACCENT	Latin-2
001	222	13/14	LATIN CAPITAL LETTER T WITH CEDILLA	Latin-2
001	224	14/00	LATIN SMALL LETTER r WITH ACUTE ACCENT	Latin-2
001	227	14/03	LATIN SMALL LETTER a WITH BREVE	Latin-2
001	229	14/05	LATIN SMALL LETTER l WITH ACUTE ACCENT	Latin-2
001	230	14/06	LATIN SMALL LETTER c WITH ACUTE ACCENT	Latin-2
001	232	14/08	LATIN SMALL LETTER c WITH CARON	Latin-2
001	234	14/10	LATIN SMALL LETTER e WITH OGONEK	Latin-2
001	236	14/12	LATIN SMALL LETTER e WITH CARON	Latin-2
001	239	14/15	LATIN SMALL LETTER d WITH CARON	Latin-2
001	240	15/00	LATIN SMALL LETTER d WITH STROKE	Latin-2
001	241	15/01	LATIN SMALL LETTER n WITH ACUTE ACCENT	Latin-2
001	242	15/02	LATIN SMALL LETTER n WITH CARON	Latin-2
001	245	15/05	LATIN SMALL LETTER o WITH DOUBLE ACUTE ACCENT	Latin-2
001	248	15/08	LATIN SMALL LETTER r WITH CARON	Latin-2
001	249	15/09	LATIN SMALL LETTER u WITH RING ABOVE	Latin-2
001	251	15/11	LATIN SMALL LETTER u WITH DOUBLE ACUTE ACCENT	Latin-2
001	254	15/14	LATIN SMALL LETTER t WITH CEDILLA	Latin-2
001	255	15/15	DOT ABOVE	Latin-2
002	161	10/01	LATIN CAPITAL LETTER H WITH STROKE	Latin-3
002	166	10/06	LATIN CAPITAL LETTER H WITH CIRCUMFLEX ACCENT	Latin-3
002	169	10/09	LATIN CAPITAL LETTER I WITH DOT ABOVE	Latin-3
002	171	10/11	LATIN CAPITAL LETTER G WITH BREVE	Latin-3
002	172	10/12	LATIN CAPITAL LETTER J WITH CIRCUMFLEX ACCENT	Latin-3
002	177	11/01	LATIN SMALL LETTER h WITH STROKE	Latin-3
002	182	11/06	LATIN SMALL LETTER h WITH CIRCUMFLEX ACCENT	Latin-3
002	185	11/09	SMALL DOTLESS LETTER i	Latin-3
002	187	11/11	LATIN SMALL LETTER g WITH BREVE	Latin-3
002	188	11/12	LATIN SMALL LETTER j WITH CIRCUMFLEX ACCENT	Latin-3
002	197	12/05	LATIN CAPITAL LETTER C WITH DOT ABOVE	Latin-3
002	198	12/06	LATIN CAPITAL LETTER C WITH CIRCUMFLEX ACCENT	Latin-3

Byte 3	Byte 4	Code Position	Name	Set
002	213	13/05	LATIN CAPITAL LETTER G WITH DOT ABOVE	Latin-3
002	216	13/08	LATIN CAPITAL LETTER G WITH CIRCUMFLEX ACCENT	Latin-3
002	221	13/13	LATIN CAPITAL LETTER U WITH BREVE	Latin-3
002	222	13/14	LATIN CAPITAL LETTER S WITH CIRCUMFLEX ACCENT	Latin-3
002	229	14/05	LATIN SMALL LETTER c WITH DOT ABOVE	Latin-3
002	230	14/06	LATIN SMALL LETTER c WITH CIRCUMFLEX ACCENT	Latin-3
002	245	15/05	LATIN SMALL LETTER g WITH DOT ABOVE	Latin-3
002	248	15/08	LATIN SMALL LETTER g WITH CIRCUMFLEX ACCENT	Latin-3
002	253	15/13	LATIN SMALL LETTER u WITH BREVE	Latin-3
002	254	15/14	LATIN SMALL LETTER s WITH CIRCUMFLEX ACCENT	Latin-3
003	162	10/02	SMALL GREENLANDIC LETTER KRA	Latin-4
003	163	10/03	LATIN CAPITAL LETTER R WITH CEDILLA	Latin-4
003	165	10/05	LATIN CAPITAL LETTER I WITH TILDE	Latin-4
003	166	10/06	LATIN CAPITAL LETTER L WITH CEDILLA	Latin-4
003	170	10/10	LATIN CAPITAL LETTER E WITH MACRON	Latin-4
003	171	10/11	LATIN CAPITAL LETTER G WITH CEDILLA	Latin-4
003	172	10/12	LATIN CAPITAL LETTER T WITH OBLIQUE STROKE	Latin-4
003	179	11/03	LATIN SMALL LETTER r WITH CEDILLA	Latin-4
003	181	11/05	LATIN SMALL LETTER i WITH TILDE	Latin-4
003	182	11/06	LATIN SMALL LETTER l WITH CEDILLA	Latin-4
003	186	11/10	LATIN SMALL LETTER e WITH MACRON	Latin-4
003	187	11/11	LATIN SMALL LETTER g WITH CEDILLA ABOVE	Latin-4
003	188	11/12	LATIN SMALL LETTER t WITH OBLIQUE STROKE	Latin-4
003	189	11/13	LAPPISH CAPITAL LETTER ENG	Latin-4
003	191	11/15	LAPPISH SMALL LETTER ENG	Latin-4
003	192	12/00	LATIN CAPITAL LETTER A WITH MACRON	Latin-4
003	199	12/07	LATIN CAPITAL LETTER I WITH OGONEK	Latin-4
003	204	12/12	LATIN CAPITAL LETTER E WITH DOT ABOVE	Latin-4
003	207	12/15	LATIN CAPITAL LETTER I WITH MACRON	Latin-4
003	209	13/01	LATIN CAPITAL LETTER N WITH CEDILLA	Latin-4
003	210	13/02	LATIN CAPITAL LETTER O WITH MACRON	Latin-4
003	211	13/03	LATIN CAPITAL LETTER K WITH CEDILLA	Latin-4
003	217	13/09	LATIN CAPITAL LETTER U WITH OGONEK	Latin-4
003	221	13/13	LATIN CAPITAL LETTER U WITH TILDE	Latin-4
003	222	13/14	LATIN CAPITAL LETTER U WITH MACRON	Latin-4
003	224	14/00	LATIN SMALL LETTER a WITH MACRON	Latin-4
003	231	14/07	LATIN SMALL LETTER i WITH OGONEK	Latin-4
003	236	14/12	LATIN SMALL LETTER e WITH DOT ABOVE	Latin-4
003	239	14/15	LATIN SMALL LETTER i WITH MACRON	Latin-4
003	241	15/01	LATIN SMALL LETTER n WITH CEDILLA	Latin-4
003	242	15/02	LATIN SMALL LETTER o WITH MACRON	Latin-4
003	243	15/03	LATIN SMALL LETTER k WITH CEDILLA	Latin-4
003	249	15/09	LATIN SMALL LETTER u WITH OGONEK	Latin-4
003	253	15/13	LATIN SMALL LETTER u WITH TILDE	Latin-4
003	254	15/14	LATIN SMALL LETTER u WITH MACRON	Latin-4
004	126	07/14	OVERLINE	Kana
004	161	10/01	KANA FULL STOP	Kana
004	162	10/02	KANA OPENING BRACKET	Kana
004	163	10/03	KANA CLOSING BRACKET	Kana
004	164	10/04	KANA COMMA	Kana
004	165	10/05	KANA CONJUNCTIVE	Kana
004	166	10/06	KANA LETTER WO	Kana

Byte 3	Byte 4	Code Position	Name	Set
004	167	10/07	KANA LETTER SMALL A	Kana
004	168	10/08	KANA LETTER SMALL I	Kana
004	169	10/09	KANA LETTER SMALL U	Kana
004	170	10/10	KANA LETTER SMALL E	Kana
004	171	10/11	KANA LETTER SMALL O	Kana
004	172	10/12	KANA LETTER SMALL YA	Kana
004	173	10/13	KANA LETTER SMALL YU	Kana
004	174	10/14	KANA LETTER SMALL YO	Kana
004	175	10/15	KANA LETTER SMALL TSU	Kana
004	176	11/00	PROLONGED SOUND SYMBOL	Kana
004	177	11/01	KANA LETTER A	Kana
004	178	11/02	KANA LETTER I	Kana
004	179	11/03	KANA LETTER U	Kana
004	180	11/04	KANA LETTER E	Kana
004	181	11/05	KANA LETTER O	Kana
004	182	11/06	KANA LETTER KA	Kana
004	183	11/07	KANA LETTER KI	Kana
004	184	11/08	KANA LETTER KU	Kana
004	185	11/09	KANA LETTER KE	Kana
004	186	11/10	KANA LETTER KO	Kana
004	187	11/11	KANA LETTER SA	Kana
004	188	11/12	KANA LETTER SHI	Kana
004	189	11/13	KANA LETTER SU	Kana
004	190	11/14	KANA LETTER SE	Kana
004	191	11/15	KANA LETTER SO	Kana
004	192	12/00	KANA LETTER TA	Kana
004	193	12/01	KANA LETTER CHI	Kana
004	194	12/02	KANA LETTER TSU	Kana
004	195	12/03	KANA LETTER TE	Kana
004	196	12/04	KANA LETTER TO	Kana
004	197	12/05	KANA LETTER NA	Kana
004	198	12/06	KANA LETTER NI	Kana
004	199	12/07	KANA LETTER NU	Kana
004	200	12/08	KANA LETTER NE	Kana
004	201	12/09	KANA LETTER NO	Kana
004	202	12/10	KANA LETTER HA	Kana
004	203	12/11	KANA LETTER HI	Kana
004	204	12/12	KANA LETTER FU	Kana
004	205	12/13	KANA LETTER HE	Kana
004	206	12/14	KANA LETTER HO	Kana
004	207	12/15	KANA LETTER MA	Kana
004	208	13/00	KANA LETTER MI	Kana
004	209	13/01	KANA LETTER MU	Kana
004	210	13/02	KANA LETTER ME	Kana
004	211	13/03	KANA LETTER MO	Kana
004	212	13/04	KANA LETTER YA	Kana
004	213	13/05	KANA LETTER YU	Kana
004	214	13/06	KANA LETTER YO	Kana
004	215	13/07	KANA LETTER RA	Kana
004	216	13/08	KANA LETTER RI	Kana
004	217	13/09	KANA LETTER RU	Kana
004	218	13/10	KANA LETTER RE	Kana

Byte 3	Byte 4	Code Position	Name	Set
004	219	13/11	KANA LETTER RO	Kana
004	220	13/12	KANA LETTER WA	Kana
004	221	13/13	KANA LETTER N	Kana
004	222	13/14	VOICED SOUND SYMBOL	Kana
004	223	13/15	SEMIVOICED SOUND SYMBOL	Kana
005	172	10/12	ARABIC COMMA	Arabic
005	187	11/11	ARABIC SEMICOLON	Arabic
005	191	11/15	ARABIC QUESTION MARK	Arabic
005	193	12/01	ARABIC LETTER HAMZA	Arabic
005	194	12/02	ARABIC LETTER MADDA ON ALEF	Arabic
005	195	12/03	ARABIC LETTER HAMZA ON ALEF	Arabic
005	196	12/04	ARABIC LETTER HAMZA ON WAW	Arabic
005	197	12/05	ARABIC LETTER HAMZA UNDER ALEF	Arabic
005	198	12/06	ARABIC LETTER HAMZA ON YEH	Arabic
005	199	12/07	ARABIC LETTER ALEF	Arabic
005	200	12/08	ARABIC LETTER BEH	Arabic
005	201	12/09	ARABIC LETTER TEH MARBUTA	Arabic
005	202	12/10	ARABIC LETTER TEH	Arabic
005	203	12/11	ARABIC LETTER THEH	Arabic
005	204	12/12	ARABIC LETTER JEEM	Arabic
005	205	12/13	ARABIC LETTER HAH	Arabic
005	206	12/14	ARABIC LETTER KHAH	Arabic
005	207	12/15	ARABIC LETTER DAL	Arabic
005	208	13/00	ARABIC LETTER THAL	Arabic
005	209	13/01	ARABIC LETTER RA	Arabic
005	210	13/02	ARABIC LETTER ZAIN	Arabic
005	211	13/03	ARABIC LETTER SEEN	Arabic
005	212	13/04	ARABIC LETTER SHEEN	Arabic
005	213	13/05	ARABIC LETTER SAD	Arabic
005	214	13/06	ARABIC LETTER DAD	Arabic
005	215	13/07	ARABIC LETTER TAH	Arabic
005	216	13/08	ARABIC LETTER ZAH	Arabic
005	217	13/09	ARABIC LETTER AIN	Arabic
005	218	13/10	ARABIC LETTER GHAIN	Arabic
005	224	14/00	ARABIC LETTER TATWEEL	Arabic
005	225	14/01	ARABIC LETTER FEH	Arabic
005	226	14/02	ARABIC LETTER QAF	Arabic
005	227	14/03	ARABIC LETTER KAF	Arabic
005	228	14/04	ARABIC LETTER LAM	Arabic
005	229	14/05	ARABIC LETTER MEEM	Arabic
005	230	14/06	ARABIC LETTER NOON	Arabic
005	231	14/07	ARABIC LETTER HA	Arabic
005	232	14/08	ARABIC LETTER WAW	Arabic
005	233	14/09	ARABIC LETTER ALEF MAKSURA	Arabic
005	234	14/10	ARABIC LETTER YEH	Arabic
005	235	14/11	ARABIC LETTER FATHATAN	Arabic
005	236	14/12	ARABIC LETTER DAMMATAN	Arabic
005	237	14/13	ARABIC LETTER KASRATAN	Arabic
005	238	14/14	ARABIC LETTER FATHA	Arabic
005	239	14/15	ARABIC LETTER DAMMA	Arabic
005	240	15/00	ARABIC LETTER KASRA	Arabic
005	241	15/01	ARABIC LETTER SHADDA	Arabic

Byte 3	Byte 4	Code Position	Name	Set
005	242	15/02	ARABIC LETTER SUKUN	Arabic
006	161	10/01	SERBOCROATIAN CYRILLIC SMALL LETTER DJE	Cyrillic
006	162	10/02	MACEDONIAN CYRILLIC SMALL LETTER GJE	Cyrillic
006	163	10/03	CYRILLIC SMALL LETTER IO	Cyrillic
006	164	10/04	UKRAINIAN CYRILLIC SMALL LETTER IE	Cyrillic
006	165	10/05	MACEDONIAN SMALL LETTER DSE	Cyrillic
006	166	10/06	BYELORUSSIAN/UKRAINIAN CYRILLIC SMALL LETTER I	Cyrillic
006	167	10/07	UKRAINIAN SMALL LETTER YI	Cyrillic
006	168	10/08	CYRILLIC SMALL LETTER JE	Cyrillic
006	169	10/09	CYRILLIC SMALL LETTER LJE	Cyrillic
006	170	10/10	CYRILLIC SMALL LETTER NJE	Cyrillic
006	171	10/11	SERBIAN SMALL LETTER TSHE	Cyrillic
006	172	10/12	MACEDONIAN CYRILLIC SMALL LETTER KJE	Cyrillic
006	174	10/14	BYELORUSSIAN SMALL LETTER SHORT U	Cyrillic
006	175	10/15	CYRILLIC SMALL LETTER DZHE	Cyrillic
006	176	11/00	NUMERO SIGN	Cyrillic
006	177	11/01	SERBOCROATIAN CYRILLIC CAPITAL LETTER DJE	Cyrillic
006	178	11/02	MACEDONIAN CYRILLIC CAPITAL LETTER GJE	Cyrillic
006	179	11/03	CYRILLIC CAPITAL LETTER IO	Cyrillic
006	180	11/04	UKRAINIAN CYRILLIC CAPITAL LETTER IE	Cyrillic
006	181	11/05	MACEDONIAN CAPITAL LETTER DSE	Cyrillic
006	182	11/06	BYELORUSSIAN/UKRAINIAN CYRILLIC CAPITAL LETTER I	Cyrillic
006	183	11/07	UKRAINIAN CAPITAL LETTER YI	Cyrillic
006	184	11/08	CYRILLIC CAPITAL LETTER JE	Cyrillic
006	185	11/09	CYRILLIC CAPITAL LETTER LJE	Cyrillic
006	186	11/10	CYRILLIC CAPITAL LETTER NJE	Cyrillic
006	187	11/11	SERBIAN CAPITAL LETTER TSHE	Cyrillic
006	188	11/12	MACEDONIAN CYRILLIC CAPITAL LETTER KJE	Cyrillic
006	190	11/14	BYELORUSSIAN CAPITAL LETTER SHORT U	Cyrillic
006	191	11/15	CYRILLIC CAPITAL LETTER DZHE	Cyrillic
006	192	12/00	CYRILLIC SMALL LETTER YU	Cyrillic
006	193	12/01	CYRILLIC SMALL LETTER A	Cyrillic
006	194	12/02	CYRILLIC SMALL LETTER BE	Cyrillic
006	195	12/03	CYRILLIC SMALL LETTER TSE	Cyrillic
006	196	12/04	CYRILLIC SMALL LETTER DE	Cyrillic
006	197	12/05	CYRILLIC SMALL LETTER IE	Cyrillic
006	198	12/06	CYRILLIC SMALL LETTER EF	Cyrillic
006	199	12/07	CYRILLIC SMALL LETTER GHE	Cyrillic
006	200	12/08	CYRILLIC SMALL LETTER HA	Cyrillic
006	201	12/09	CYRILLIC SMALL LETTER I	Cyrillic
006	202	12/10	CYRILLIC SMALL LETTER SHORT I	Cyrillic
006	203	12/11	CYRILLIC SMALL LETTER KA	Cyrillic
006	204	12/12	CYRILLIC SMALL LETTER EL	Cyrillic
006	205	12/13	CYRILLIC SMALL LETTER EM	Cyrillic
006	206	12/14	CYRILLIC SMALL LETTER EN	Cyrillic
006	207	12/15	CYRILLIC SMALL LETTER O	Cyrillic
006	208	13/00	CYRILLIC SMALL LETTER PE	Cyrillic
006	209	13/01	CYRILLIC SMALL LETTER YA	Cyrillic
006	210	13/02	CYRILLIC SMALL LETTER ER	Cyrillic
006	211	13/03	CYRILLIC SMALL LETTER ES	Cyrillic
006	212	13/04	CYRILLIC SMALL LETTER TE	Cyrillic
006	213	13/05	CYRILLIC SMALL LETTER U	Cyrillic

Byte 3	Byte 4	Code Position	Name	Set
006	214	13/06	CYRILLIC SMALL LETTER ZHE	Cyrillic
006	215	13/07	CYRILLIC SMALL LETTER VE	Cyrillic
006	216	13/08	CYRILLIC SMALL SOFT SIGN	Cyrillic
006	217	13/09	CYRILLIC SMALL LETTER YERU	Cyrillic
006	218	13/10	CYRILLIC SMALL LETTER ZE	Cyrillic
006	219	13/11	CYRILLIC SMALL LETTER SHA	Cyrillic
006	220	13/12	CYRILLIC SMALL LETTER E	Cyrillic
006	221	13/13	CYRILLIC SMALL LETTER SHCHA	Cyrillic
006	222	13/14	CYRILLIC SMALL LETTER CHE	Cyrillic
006	223	13/15	CYRILLIC SMALL HARD SIGN	Cyrillic
006	224	14/00	CYRILLIC CAPITAL LETTER YU	Cyrillic
006	225	14/01	CYRILLIC CAPITAL LETTER A	Cyrillic
006	226	14/02	CYRILLIC CAPITAL LETTER BE	Cyrillic
006	227	14/03	CYRILLIC CAPITAL LETTER TSE	Cyrillic
006	228	14/04	CYRILLIC CAPITAL LETTER DE	Cyrillic
006	229	14/05	CYRILLIC CAPITAL LETTER IE	Cyrillic
006	230	14/06	CYRILLIC CAPITAL LETTER EF	Cyrillic
006	231	14/07	CYRILLIC CAPITAL LETTER GHE	Cyrillic
006	232	14/08	CYRILLIC CAPITAL LETTER HA	Cyrillic
006	233	14/09	CYRILLIC CAPITAL LETTER I	Cyrillic
006	234	14/10	CYRILLIC CAPITAL LETTER SHORT I	Cyrillic
006	235	14/11	CYRILLIC CAPITAL LETTER KA	Cyrillic
006	236	14/12	CYRILLIC CAPITAL LETTER EL	Cyrillic
006	237	14/13	CYRILLIC CAPITAL LETTER EM	Cyrillic
006	238	14/14	CYRILLIC CAPITAL LETTER EN	Cyrillic
006	239	14/15	CYRILLIC CAPITAL LETTER O	Cyrillic
006	240	15/00	CYRILLIC CAPITAL LETTER PE	Cyrillic
006	241	15/01	CYRILLIC CAPITAL LETTER YA	Cyrillic
006	242	15/02	CYRILLIC CAPITAL LETTER ER	Cyrillic
006	243	15/03	CYRILLIC CAPITAL LETTER ES	Cyrillic
006	244	15/04	CYRILLIC CAPITAL LETTER TE	Cyrillic
006	245	15/05	CYRILLIC CAPITAL LETTER U	Cyrillic
006	246	15/06	CYRILLIC CAPITAL LETTER ZHE	Cyrillic
006	247	15/07	CYRILLIC CAPITAL LETTER VE	Cyrillic
006	248	15/08	CYRILLIC CAPITAL SOFT SIGN	Cyrillic
006	249	15/09	CYRILLIC CAPITAL LETTER YERU	Cyrillic
006	250	15/10	CYRILLIC CAPITAL LETTER ZE	Cyrillic
006	251	15/11	CYRILLIC CAPITAL LETTER SHA	Cyrillic
006	252	15/12	CYRILLIC CAPITAL LETTER E	Cyrillic
006	253	15/13	CYRILLIC CAPITAL LETTER SHCHA	Cyrillic
006	254	15/14	CYRILLIC CAPITAL LETTER CHE	Cyrillic
006	255	15/15	CYRILLIC CAPITAL HARD SIGN	Cyrillic
007	161	10/01	GREEK CAPITAL LETTER ALPHA WITH ACCENT	Greek
007	162	10/02	GREEK CAPITAL LETTER EPSILON WITH ACCENT	Greek
007	163	10/03	GREEK CAPITAL LETTER ETA WITH ACCENT	Greek
007	164	10/04	GREEK CAPITAL LETTER IOTA WITH ACCENT	Greek
007	165	10/05	GREEK CAPITAL LETTER IOTA WITH DIAERESIS	Greek
007	167	10/07	GREEK CAPITAL LETTER OMICRON WITH ACCENT	Greek
007	168	10/08	GREEK CAPITAL LETTER UPSILON WITH ACCENT	Greek
007	169	10/09	GREEK CAPITAL LETTER UPSILON WITH DIAERESIS	Greek
007	171	10/11	GREEK CAPITAL LETTER OMEGA WITH ACCENT	Greek
007	174	10/14	DIAERESIS AND ACCENT	Greek

Byte 3	Byte 4	Code Position	Name	Set
007	175	10/15	HORIZONTAL BAR	Greek
007	177	11/01	GREEK SMALL LETTER ALPHA WITH ACCENT	Greek
007	178	11/02	GREEK SMALL LETTER EPSILON WITH ACCENT	Greek
007	179	11/03	GREEK SMALL LETTER ETA WITH ACCENT	Greek
007	180	11/04	GREEK SMALL LETTER IOTA WITH ACCENT	Greek
007	181	11/05	GREEK SMALL LETTER IOTA WITH DIAERESIS	Greek
007	182	11/06	GREEK SMALL LETTER IOTA WITH ACCENT+DIAERESIS	Greek
007	183	11/07	GREEK SMALL LETTER OMICRON WITH ACCENT	Greek
007	184	11/08	GREEK SMALL LETTER UPSILON WITH ACCENT	Greek
007	185	11/09	GREEK SMALL LETTER UPSILON WITH DIAERESIS	Greek
007	186	11/10	GREEK SMALL LETTER UPSILON WITH ACCENT+DIAERESIS	Greek
007	187	11/11	GREEK SMALL LETTER OMEGA WITH ACCENT	Greek
007	193	12/01	GREEK CAPITAL LETTER ALPHA	Greek
007	194	12/02	GREEK CAPITAL LETTER BETA	Greek
007	195	12/03	GREEK CAPITAL LETTER GAMMA	Greek
007	196	12/04	GREEK CAPITAL LETTER DELTA	Greek
007	197	12/05	GREEK CAPITAL LETTER EPSILON	Greek
007	198	12/06	GREEK CAPITAL LETTER ZETA	Greek
007	199	12/07	GREEK CAPITAL LETTER ETA	Greek
007	200	12/08	GREEK CAPITAL LETTER THETA	Greek
007	201	12/09	GREEK CAPITAL LETTER IOTA	Greek
007	202	12/10	GREEK CAPITAL LETTER KAPPA	Greek
007	203	12/11	GREEK CAPITAL LETTER LAMDA	Greek
007	204	12/12	GREEK CAPITAL LETTER MU	Greek
007	205	12/13	GREEK CAPITAL LETTER NU	Greek
007	206	12/14	GREEK CAPITAL LETTER XI	Greek
007	207	12/15	GREEK CAPITAL LETTER OMICRON	Greek
007	208	13/00	GREEK CAPITAL LETTER PI	Greek
007	209	13/01	GREEK CAPITAL LETTER RHO	Greek
007	210	13/02	GREEK CAPITAL LETTER SIGMA	Greek
007	212	13/04	GREEK CAPITAL LETTER TAU	Greek
007	213	13/05	GREEK CAPITAL LETTER UPSILON	Greek
007	214	13/06	GREEK CAPITAL LETTER PHI	Greek
007	215	13/07	GREEK CAPITAL LETTER CHI	Greek
007	216	13/08	GREEK CAPITAL LETTER PSI	Greek
007	217	13/09	GREEK CAPITAL LETTER OMEGA	Greek
007	225	14/01	GREEK SMALL LETTER ALPHA	Greek
007	226	14/02	GREEK SMALL LETTER BETA	Greek
007	227	14/03	GREEK SMALL LETTER GAMMA	Greek
007	228	14/04	GREEK SMALL LETTER DELTA	Greek
007	229	14/05	GREEK SMALL LETTER EPSILON	Greek
007	230	14/06	GREEK SMALL LETTER ZETA	Greek
007	231	14/07	GREEK SMALL LETTER ETA	Greek
007	232	14/08	GREEK SMALL LETTER THETA	Greek
007	233	14/09	GREEK SMALL LETTER IOTA	Greek
007	234	14/10	GREEK SMALL LETTER KAPPA	Greek
007	235	14/11	GREEK SMALL LETTER LAMDA	Greek
007	236	14/12	GREEK SMALL LETTER MU	Greek
007	237	14/13	GREEK SMALL LETTER NU	Greek
007	238	14/14	GREEK SMALL LETTER XI	Greek
007	239	14/15	GREEK SMALL LETTER OMICRON	Greek
007	240	15/00	GREEK SMALL LETTER PI	Greek

Byte 3	Byte 4	Code Position	Name	Set
007	241	15/01	GREEK SMALL LETTER RHO	Greek
007	242	15/02	GREEK SMALL LETTER SIGMA	Greek
007	243	15/03	GREEK SMALL LETTER FINAL SMALL SIGMA	Greek
007	244	15/04	GREEK SMALL LETTER TAU	Greek
007	245	15/05	GREEK SMALL LETTER UPSILON	Greek
007	246	15/06	GREEK SMALL LETTER PHI	Greek
007	247	15/07	GREEK SMALL LETTER CHI	Greek
007	248	15/08	GREEK SMALL LETTER PSI	Greek
007	249	15/09	GREEK SMALL LETTER OMEGA	Greek
008	161	10/01	LEFT RADICAL	Technical
008	162	10/02	TOP LEFT RADICAL	Technical
008	163	10/03	HORIZONTAL CONNECTOR	Technical
008	164	10/04	TOP INTEGRAL	Technical
008	165	10/05	BOTTOM INTEGRAL	Technical
008	166	10/06	VERTICAL CONNECTOR	Technical
008	167	10/07	TOP LEFT SQUARE BRACKET	Technical
008	168	10/08	BOTTOM LEFT SQUARE BRACKET	Technical
008	169	10/09	TOP RIGHT SQUARE BRACKET	Technical
008	170	10/10	BOTTOM RIGHT SQUARE BRACKET	Technical
008	171	10/11	TOP LEFT PARENTHESIS	Technical
008	172	10/12	BOTTOM LEFT PARENTHESIS	Technical
008	173	10/13	TOP RIGHT PARENTHESIS	Technical
008	174	10/14	BOTTOM RIGHT PARENTHESIS	Technical
008	175	10/15	LEFT MIDDLE CURLY BRACE	Technical
008	176	11/00	RIGHT MIDDLE CURLY BRACE	Technical
008	177	11/01	TOP LEFT SUMMATION	Technical
008	178	11/02	BOTTOM LEFT SUMMATION	Technical
008	179	11/03	TOP VERTICAL SUMMATION CONNECTOR	Technical
008	180	11/04	BOTTOM VERTICAL SUMMATION CONNECTOR	Technical
008	181	11/05	TOP RIGHT SUMMATION	Technical
008	182	11/06	BOTTOM RIGHT SUMMATION	Technical
008	183	11/07	RIGHT MIDDLE SUMMATION	Technical
008	188	11/12	LESS THAN OR EQUAL SIGN	Technical
008	189	11/13	NOT EQUAL SIGN	Technical
008	190	11/14	GREATER THAN OR EQUAL SIGN	Technical
008	191	11/15	INTEGRAL	Technical
008	192	12/00	THEREFORE	Technical
008	193	12/01	VARIATION, PROPORTIONAL TO	Technical
008	194	12/02	INFINITY	Technical
008	197	12/05	NABLA, DEL	Technical
008	200	12/08	IS APPROXIMATE TO	Technical
008	201	12/09	SIMILAR OR EQUAL TO	Technical
008	205	12/13	IF AND ONLY IF	Technical
008	206	12/14	IMPLIES	Technical
008	207	12/15	IDENTICAL TO	Technical
008	214	13/06	RADICAL	Technical
008	218	13/10	IS INCLUDED IN	Technical
008	219	13/11	INCLUDES	Technical
008	220	13/12	INTERSECTION	Technical
008	221	13/13	UNION	Technical
008	222	13/14	LOGICAL AND	Technical
008	223	13/15	LOGICAL OR	Technical

Byte 3	Byte 4	Code Position	Name	Set
008	239	14/15	PARTIAL DERIVATIVE	Technical
008	246	15/06	FUNCTION	Technical
008	251	15/11	LEFT ARROW	Technical
008	252	15/12	UPWARD ARROW	Technical
008	253	15/13	RIGHT ARROW	Technical
008	254	15/14	DOWNWARD ARROW	Technical
009	223	13/15	BLANK	Special
009	224	14/00	SOLID DIAMOND	Special
009	225	14/01	CHECKERBOARD	Special
009	226	14/02	‘‘HT’’	Special
009	227	14/03	‘‘FF’’	Special
009	228	14/04	‘‘CR’’	Special
009	229	14/05	‘‘LF’’	Special
009	232	14/08	‘‘NL’’	Special
009	233	14/09	‘‘VT’’	Special
009	234	14/10	LOWER-RIGHT CORNER	Special
009	235	14/11	UPPER-RIGHT CORNER	Special
009	236	14/12	UPPER-LEFT CORNER	Special
009	237	14/13	LOWER-LEFT CORNER	Special
009	238	14/14	CROSSING-LINES	Special
009	239	14/15	HORIZONTAL LINE, SCAN 1	Special
009	240	15/00	HORIZONTAL LINE, SCAN 3	Special
009	241	15/01	HORIZONTAL LINE, SCAN 5	Special
009	242	15/02	HORIZONTAL LINE, SCAN 7	Special
009	243	15/03	HORIZONTAL LINE, SCAN 9	Special
009	244	15/04	LEFT ‘‘T’’	Special
009	245	15/05	RIGHT ‘‘T’’	Special
009	246	15/06	BOTTOM ‘‘T’’	Special
009	247	15/07	TOP ‘‘T’’	Special
009	248	15/08	VERTICAL BAR	Special
010	161	10/01	EM SPACE	Publish
010	162	10/02	EN SPACE	Publish
010	163	10/03	3/EM SPACE	Publish
010	164	10/04	4/EM SPACE	Publish
010	165	10/05	DIGIT SPACE	Publish
010	166	10/06	PUNCTUATION SPACE	Publish
010	167	10/07	THIN SPACE	Publish
010	168	10/08	HAIR SPACE	Publish
010	169	10/09	EM DASH	Publish
010	170	10/10	EN DASH	Publish
010	172	10/12	SIGNIFICANT BLANK SYMBOL	Publish
010	174	10/14	ELLIPSIS	Publish
010	175	10/15	DOUBLE BASELINE DOT	Publish
010	176	11/00	VULGAR FRACTION ONE THIRD	Publish
010	177	11/01	VULGAR FRACTION TWO THIRDS	Publish
010	178	11/02	VULGAR FRACTION ONE FIFTH	Publish
010	179	11/03	VULGAR FRACTION TWO FIFTHS	Publish
010	180	11/04	VULGAR FRACTION THREE FIFTHS	Publish
010	181	11/05	VULGAR FRACTION FOUR FIFTHS	Publish
010	182	11/06	VULGAR FRACTION ONE SIXTH	Publish
010	183	11/07	VULGAR FRACTION FIVE SIXTHS	Publish
010	184	11/08	CARE OF	Publish

Byte 3	Byte 4	Code Position	Name	Set
010	187	11/11	FIGURE DASH	Publish
010	188	11/12	LEFT ANGLE BRACKET	Publish
010	189	11/13	DECIMAL POINT	Publish
010	190	11/14	RIGHT ANGLE BRACKET	Publish
010	191	11/15	MARKER	Publish
010	195	12/03	VULGAR FRACTION ONE EIGHTH	Publish
010	196	12/04	VULGAR FRACTION THREE EIGHTHS	Publish
010	197	12/05	VULGAR FRACTION FIVE EIGHTHS	Publish
010	198	12/06	VULGAR FRACTION SEVEN EIGHTHS	Publish
010	201	12/09	TRADEMARK SIGN	Publish
010	202	12/10	SIGNATURE MARK	Publish
010	203	12/11	TRADEMARK SIGN IN CIRCLE	Publish
010	204	12/12	LEFT OPEN TRIANGLE	Publish
010	205	12/13	RIGHT OPEN TRIANGLE	Publish
010	206	12/14	EM OPEN CIRCLE	Publish
010	207	12/15	EM OPEN RECTANGLE	Publish
010	208	13/00	LEFT SINGLE QUOTATION MARK	Publish
010	209	13/01	RIGHT SINGLE QUOTATION MARK	Publish
010	210	13/02	LEFT DOUBLE QUOTATION MARK	Publish
010	211	13/03	RIGHT DOUBLE QUOTATION MARK	Publish
010	212	13/04	PRESCRIPTION, TAKE, RECIPE	Publish
010	214	13/06	MINUTES	Publish
010	215	13/07	SECONDS	Publish
010	217	13/09	LATIN CROSS	Publish
010	218	13/10	HEXAGRAM	Publish
010	219	13/11	FILLED RECTANGLE BULLET	Publish
010	220	13/12	FILLED LEFT TRIANGLE BULLET	Publish
010	221	13/13	FILLED RIGHT TRIANGLE BULLET	Publish
010	222	13/14	EM FILLED CIRCLE	Publish
010	223	13/15	EM FILLED RECTANGLE	Publish
010	224	14/00	EN OPEN CIRCLE BULLET	Publish
010	225	14/01	EN OPEN SQUARE BULLET	Publish
010	226	14/02	OPEN RECTANGULAR BULLET	Publish
010	227	14/03	OPEN TRIANGULAR BULLET UP	Publish
010	228	14/04	OPEN TRIANGULAR BULLET DOWN	Publish
010	229	14/05	OPEN STAR	Publish
010	230	14/06	EN FILLED CIRCLE BULLET	Publish
010	231	14/07	EN FILLED SQUARE BULLET	Publish
010	232	14/08	FILLED TRIANGULAR BULLET UP	Publish
010	233	14/09	FILLED TRIANGULAR BULLET DOWN	Publish
010	234	14/10	LEFT POINTER	Publish
010	235	14/11	RIGHT POINTER	Publish
010	236	14/12	CLUB	Publish
010	237	14/13	DIAMOND	Publish
010	238	14/14	HEART	Publish
010	240	15/00	MALTESE CROSS	Publish
010	241	15/01	DAGGER	Publish
010	242	15/02	DOUBLE DAGGER	Publish
010	243	15/03	CHECK MARK, TICK	Publish
010	244	15/04	BALLOT CROSS	Publish
010	245	15/05	MUSICAL SHARP	Publish
010	246	15/06	MUSICAL FLAT	Publish

Byte 3	Byte 4	Code Position	Name	Set
010	247	15/07	MALE SYMBOL	Publish
010	248	15/08	FEMALE SYMBOL	Publish
010	249	15/09	TELEPHONE SYMBOL	Publish
010	250	15/10	TELEPHONE RECORDER SYMBOL	Publish
010	251	15/11	PHONOGRAPH COPYRIGHT SIGN	Publish
010	252	15/12	CARET	Publish
010	253	15/13	SINGLE LOW QUOTATION MARK	Publish
010	254	15/14	DOUBLE LOW QUOTATION MARK	Publish
010	255	15/15	CURSOR	Publish
011	163	10/03	LEFT CARET	APL
011	166	10/06	RIGHT CARET	APL
011	168	10/08	DOWN CARET	APL
011	169	10/09	UP CARET	APL
011	192	12/00	OVERBAR	APL
011	194	12/02	DOWN TACK	APL
011	195	12/03	UP SHOE (CAP)	APL
011	196	12/04	DOWN STILE	APL
011	198	12/06	UNDERBAR	APL
011	202	12/10	JOT	APL
011	204	12/12	QUAD	APL
011	206	12/14	UP TACK	APL
011	207	12/15	CIRCLE	APL
011	211	13/03	UP STILE	APL
011	214	13/06	DOWN SHOE (CUP)	APL
011	216	13/08	RIGHT SHOE	APL
011	218	13/10	LEFT SHOE	APL
011	220	13/12	LEFT TACK	APL
011	252	15/12	RIGHT TACK	APL
012	223	13/15	DOUBLE LOW LINE	Hebrew
012	224	14/00	HEBREW LETTER ALEPH	Hebrew
012	225	14/01	HEBREW LETTER BET	Hebrew
012	226	14/02	HEBREW LETTER GIMEL	Hebrew
012	227	14/03	HEBREW LETTER DALET	Hebrew
012	228	14/04	HEBREW LETTER HE	Hebrew
012	229	14/05	HEBREW LETTER WAW	Hebrew
012	230	14/06	HEBREW LETTER ZAIN	Hebrew
012	231	14/07	HEBREW LETTER CHET	Hebrew
012	232	14/08	HEBREW LETTER TET	Hebrew
012	233	14/09	HEBREW LETTER YOD	Hebrew
012	234	14/10	HEBREW LETTER FINAL KAPH	Hebrew
012	235	14/11	HEBREW LETTER KAPH	Hebrew
012	236	14/12	HEBREW LETTER LAMED	Hebrew
012	237	14/13	HEBREW LETTER FINAL MEM	Hebrew
012	238	14/14	HEBREW LETTER MEM	Hebrew
012	239	14/15	HEBREW LETTER FINAL NUN	Hebrew
012	240	15/00	HEBREW LETTER NUN	Hebrew
012	241	15/01	HEBREW LETTER SAMECH	Hebrew
012	242	15/02	HEBREW LETTER A'YIN	Hebrew
012	243	15/03	HEBREW LETTER FINAL PE	Hebrew
012	244	15/04	HEBREW LETTER PE	Hebrew
012	245	15/05	HEBREW LETTER FINAL ZADE	Hebrew
012	246	15/06	HEBREW LETTER ZADE	Hebrew

Byte 3	Byte 4	Code Position	Name	Set
012	247	15/07	HEBREW QOPH	Hebrew
012	248	15/08	HEBREW RESH	Hebrew
012	249	15/09	HEBREW SHIN	Hebrew
012	250	15/10	HEBREW TAW	Hebrew
013	161	10/01	THAI KOKAI	Thai
013	162	10/02	THAI KHOKHAI	Thai
013	163	10/03	THAI KHOKHUAT	Thai
013	164	10/04	THAI KHOKHWAI	Thai
013	165	10/05	THAI KHOKHON	Thai
013	166	10/06	THAI KHORAKHANG	Thai
013	167	10/07	THAI NGONGU	Thai
013	168	10/08	THAI CHOCHAN	Thai
013	169	10/09	THAI CHOCHING	Thai
013	170	10/10	THAI CHOCHANG	Thai
013	171	10/11	THAI SOSO	Thai
013	172	10/12	THAI CHOCHOE	Thai
013	173	10/13	THAI YOYING	Thai
013	174	10/14	THAI DOCHADA	Thai
013	175	10/15	THAI TOPATAK	Thai
013	176	11/00	THAI THOTHAN	Thai
013	177	11/01	THAI THONANGMONTHO	Thai
013	178	11/02	THAI THOPHUTHAO	Thai
013	179	11/03	THAI NONEN	Thai
013	180	11/04	THAI DODEK	Thai
013	181	11/05	THAI TOTAO	Thai
013	182	11/06	THAI THOTHUNG	Thai
013	183	11/07	THAI THOTHAHAN	Thai
013	184	11/08	THAI THOTHONG	Thai
013	185	11/09	THAI NONU	Thai
013	186	11/10	THAI BOBAIMAI	Thai
013	187	11/11	THAI POPLA	Thai
013	188	11/12	THAI PHOPHUNG	Thai
013	189	11/13	THAI FOFA	Thai
013	190	11/14	THAI PHOPHAN	Thai
013	191	11/15	THAI FOFAN	Thai
013	192	12/00	THAI PHOSAMPHAO	Thai
013	193	12/01	THAI MOMA	Thai
013	194	12/02	THAI YOYAK	Thai
013	195	12/03	THAI RORUA	Thai
013	196	12/04	THAI RU	Thai
013	197	12/05	THAI LOLING	Thai
013	198	12/06	THAI LU	Thai
013	199	12/07	THAI WOWAEN	Thai
013	200	12/08	THAI SOSALA	Thai
013	201	12/09	THAI SORUSI	Thai
013	202	12/10	THAI SOSUA	Thai
013	203	12/11	THAI HOHIP	Thai
013	204	12/12	THAI LOCHULA	Thai
013	205	12/13	THAI OANG	Thai
013	206	12/14	THAI HONOKHUK	Thai
013	207	12/15	THAI PAIYANNOI	Thai
013	208	13/00	THAI SARAA	Thai

Byte 3	Byte 4	Code Position	Name	Set
013	209	13/01	THAI MAIHANAKAT	Thai
013	210	13/02	THAI SARAAA	Thai
013	211	13/03	THAI SARAAM	Thai
013	212	13/04	THAI SARAI	Thai
013	213	13/05	THAI SARAII	Thai
013	214	13/06	THAI SARAUE	Thai
013	215	13/07	THAI SARAUEE	Thai
013	216	13/08	THAI SARAU	Thai
013	217	13/09	THAI SARAUU	Thai
013	218	13/10	THAI PHINTHU	Thai
013	222	13/14	THAI MAIHANAKAT	Thai
013	223	13/15	THAI BAHT	Thai
013	224	14/00	THAI SARAE	Thai
013	225	14/01	THAI SARAAE	Thai
013	226	14/02	THAI SARAO	Thai
013	227	14/03	THAI SARAAIMAIMUAN	Thai
013	228	14/04	THAI SARAAIMAIMALAI	Thai
013	229	14/05	THAI LAKKHANGYAO	Thai
013	230	14/06	THAI MAIYAMOK	Thai
013	231	14/07	THAI MAITAIKHU	Thai
013	232	14/08	THAI MAIEK	Thai
013	233	14/09	THAI MAITHO	Thai
013	234	14/10	THAI MAITRI	Thai
013	235	14/11	THAI MAICHATTAWA	Thai
013	236	14/12	THAI THANTHAKHAT	Thai
013	237	14/13	THAI NIKHAHIT	Thai
013	240	15/00	THAI LEKSUN	Thai
013	241	15/01	THAI LEKNUNG	Thai
013	242	15/02	THAI LEKSONG	Thai
013	243	15/03	THAI LEKSAM	Thai
013	244	15/04	THAI LEKSI	Thai
013	245	15/05	THAI LEKHA	Thai
013	246	15/06	THAI LEKHOK	Thai
013	247	15/07	THAI LEKCHET	Thai
013	248	15/08	THAI LEKPAET	Thai
013	249	15/09	THAI LEKKAO	Thai
014	161	10/01	HANGUL KIYEOG	Korean
014	162	10/02	HANGUL SSANG KIYEOG	Korean
014	163	10/03	HANGUL KIYEOG SIOS	Korean
014	164	10/04	HANGUL NIEUN	Korean
014	165	10/05	HANGUL NIEUN JIEUJ	Korean
014	166	10/06	HANGUL NIEUN HIEUH	Korean
014	167	10/07	HANGUL DIKEUD	Korean
014	168	10/08	HANGUL SSANG DIKEUD	Korean
014	169	10/09	HANGUL RIEUL	Korean
014	170	10/10	HANGUL RIEUL KIYEOG	Korean
014	171	10/11	HANGUL RIEUL MIEUM	Korean
014	172	10/12	HANGUL RIEUL PIEUB	Korean
014	173	10/13	HANGUL RIEUL SIOS	Korean
014	174	10/14	HANGUL RIEUL TIEUT	Korean
014	175	10/15	HANGUL RIEUL PHIEUF	Korean
014	176	11/00	HANGUL RIEUL HIEUH	Korean

Byte 3	Byte 4	Code Position	Name	Set
014	177	11/01	HANGUL MIEUM	Korean
014	178	11/02	HANGUL PIEUB	Korean
014	179	11/03	HANGUL SSANG PIEUB	Korean
014	180	11/04	HANGUL PIEUB SIOS	Korean
014	181	11/05	HANGUL SIOS	Korean
014	182	11/06	HANGUL SSANG SIOS	Korean
014	183	11/07	HANGUL IEUNG	Korean
014	184	11/08	HANGUL JIEUJ	Korean
014	185	11/09	HANGUL SSANG JIEUJ	Korean
014	186	11/10	HANGUL CIEUC	Korean
014	187	11/11	HANGUL KHIEUQ	Korean
014	188	11/12	HANGUL TIEUT	Korean
014	189	11/13	HANGUL PHIEUF	Korean
014	190	11/14	HANGUL HIEUH	Korean
014	191	11/15	HANGUL A	Korean
014	192	12/00	HANGUL AE	Korean
014	193	12/01	HANGUL YA	Korean
014	194	12/02	HANGUL YAE	Korean
014	195	12/03	HANGUL EO	Korean
014	196	12/04	HANGUL E	Korean
014	197	12/05	HANGUL YEO	Korean
014	198	12/06	HANGUL YE	Korean
014	199	12/07	HANGUL O	Korean
014	200	12/08	HANGUL WA	Korean
014	201	12/09	HANGUL WAE	Korean
014	202	12/10	HANGUL OE	Korean
014	203	12/11	HANGUL YO	Korean
014	204	12/12	HANGUL U	Korean
014	205	12/13	HANGUL WEO	Korean
014	206	12/14	HANGUL WE	Korean
014	207	12/15	HANGUL WI	Korean
014	208	13/00	HANGUL YU	Korean
014	209	13/01	HANGUL EU	Korean
014	210	13/02	HANGUL YI	Korean
014	211	13/03	HANGUL I	Korean
014	212	13/04	HANGUL JONG SEONG KIYEOG	Korean
014	213	13/05	HANGUL JONG SEONG SSANG KIYEOG	Korean
014	214	13/06	HANGUL JONG SEONG KIYEOG SIOS	Korean
014	215	13/07	HANGUL JONG SEONG NIEUN	Korean
014	216	13/08	HANGUL JONG SEONG NIEUN JIEUJ	Korean
014	217	13/09	HANGUL JONG SEONG NIEUN HIEUH	Korean
014	218	13/10	HANGUL JONG SEONG DIKEUD	Korean
014	219	13/11	HANGUL JONG SEONG RIEUL	Korean
014	220	13/12	HANGUL JONG SEONG RIEUL KIYEOG	Korean
014	221	13/13	HANGUL JONG SEONG RIEUL MIEUM	Korean
014	222	13/14	HANGUL JONG SEONG RIEUL PIEUB	Korean
014	223	13/15	HANGUL JONG SEONG RIEUL SIOS	Korean
014	224	14/00	HANGUL JONG SEONG RIEUL TIEUT	Korean
014	225	14/01	HANGUL JONG SEONG RIEUL PHIEUF	Korean
014	226	14/02	HANGUL JONG SEONG RIEUL HIEUH	Korean
014	227	14/03	HANGUL JONG SEONG MIEUM	Korean
014	228	14/04	HANGUL JONG SEONG PIEUB	Korean

Byte 3	Byte 4	Code Position	Name	Set
014	229	14/05	HANGUL JONG SEONG PIEUB SIOS	Korean
014	230	14/06	HANGUL JONG SEONG SIOS	Korean
014	231	14/07	HANGUL JONG SEONG SSANG SIOS	Korean
014	232	14/08	HANGUL JONG SEONG IEUNG	Korean
014	233	14/09	HANGUL JONG SEONG JIEUJ	Korean
014	234	14/10	HANGUL JONG SEONG CIEUC	Korean
014	235	14/11	HANGUL JONG SEONG KHIEUQ	Korean
014	236	14/12	HANGUL JONG SEONG TIEUT	Korean
014	237	14/13	HANGUL JONG SEONG PHIEUF	Korean
014	238	14/14	HANGUL JONG SEONG HIEUH	Korean
014	239	14/15	HANGUL RIEUL YEORIN HIEUH	Korean
014	240	15/00	HANGUL SUNKYEONGEUM MIEUM	Korean
014	241	15/01	HANGUL SUNKYEONGEUM PIEUB	Korean
014	242	15/02	HANGUL PAN SIOS	Korean
014	243	15/03	HANGUL KKOGJI DALRIN IEUNG	Korean
014	244	15/04	HANGUL SUNKYEONGEUM PHIEUF	Korean
014	245	15/05	HANGUL YEORIN HIEUH	Korean
014	246	15/06	HANGUL ARAE A	Korean
014	247	15/07	HANGUL ARAE AE	Korean
014	248	15/08	HANGUL JONG SEONG PAN SIOS	Korean
014	249	15/09	HANGUL JONG SEONG KKOGJI DALRIN IEUNG	Korean
014	250	15/10	HANGUL JONG SEONG YEORIN HIEUH	Korean
014	255	15/15	KOREAN WON	Korean
032	160	10/00	EURO SIGN	Currency
032	161	10/01	COLON SIGN	Currency
032	162	10/02	CRUZEIRO SIGN	Currency
032	163	10/03	FRENCH FRANC SIGN	Currency
032	164	10/04	LIRA SIGN	Currency
032	165	10/05	MILL SIGN	Currency
032	166	10/06	NAIRA SIGN	Currency
032	167	10/07	PESETA SIGN	Currency
032	168	10/08	RUPEE SIGN	Currency
032	169	10/09	WON SIGN	Currency
032	170	10/10	NEW SHEQEL SIGN	Currency
032	171	10/11	DONG SIGN	Currency
255	008	00/08	BACKSPACE, BACK SPACE, BACK CHAR	Keyboard
255	009	00/09	TAB	Keyboard
255	010	00/10	LINEFEED, LF	Keyboard
255	011	00/11	CLEAR	Keyboard
255	013	00/13	RETURN, ENTER	Keyboard
255	019	01/03	PAUSE, HOLD	Keyboard
255	020	01/04	SCROLL LOCK	Keyboard
255	021	01/05	SYS REQ, SYSTEM REQUEST	Keyboard
255	027	01/11	ESCAPE	Keyboard
255	032	02/00	MULTI-KEY CHARACTER PREFACE	Keyboard
255	033	02/01	KANJI, KANJI CONVERT	Keyboard
255	034	02/02	MUHENKAN	Keyboard
255	035	02/03	HENKAN MODE	Keyboard
255	036	02/04	ROMAJI	Keyboard
255	037	02/05	HIRAGANA	Keyboard
255	038	02/06	KATAKANA	Keyboard
255	039	02/07	HIRAGANA/KATAKANA TOGGLE	Keyboard

Byte 3	Byte 4	Code Position	Name	Set
255	040	02/08	ZENKAKU	Keyboard
255	041	02/09	HANKAKU	Keyboard
255	042	02/10	ZENKAKU/HANKAKU TOGGLE	Keyboard
255	043	02/11	TOUROKU	Keyboard
255	044	02/12	MASSYO	Keyboard
255	045	02/13	KANA LOCK	Keyboard
255	046	02/14	KANA SHIFT	Keyboard
255	047	02/15	EISU SHIFT	Keyboard
255	048	03/00	EISU TOGGLE	Keyboard
255	049	03/01	HANGUL START/STOP (TOGGLE)	Keyboard
255	050	03/02	HANGUL START	Keyboard
255	051	03/03	HANGUL END, ENGLISH START	Keyboard
255	052	03/04	START HANGUL/HANJA CONVERSION	Keyboard
255	053	03/05	HANGUL JAMO MODE	Keyboard
255	054	03/06	HANGUL ROMAJA MODE	Keyboard
255	055	03/07	HANGUL CODE INPUT	Keyboard
255	056	03/08	HANGUL JEONJA MODE	Keyboard
255	057	03/09	HANGUL BANJA MODE	Keyboard
255	058	03/10	HANGUL PREHANJA CONVERSION	Keyboard
255	059	03/11	HANGUL POSTHANJA CONVERSION	Keyboard
255	060	03/12	HANGUL SINGLE CANDIDATE	Keyboard
255	061	03/13	HANGUL MULTIPLE CANDIDATE	Keyboard
255	062	03/14	HANGUL PREVIOUS CANDIDATE	Keyboard
255	063	03/15	HANGUL SPECIAL SYMBOLS	Keyboard
255	080	05/00	HOME	Keyboard
255	081	05/01	LEFT, MOVE LEFT, LEFT ARROW	Keyboard
255	082	05/02	UP, MOVE UP, UP ARROW	Keyboard
255	083	05/03	RIGHT, MOVE RIGHT, RIGHT ARROW	Keyboard
255	084	05/04	DOWN, MOVE DOWN, DOWN ARROW	Keyboard
255	085	05/05	PRIOR, PREVIOUS, PAGE UP	Keyboard
255	086	05/06	NEXT, PAGE DOWN	Keyboard
255	087	05/07	END, EOL	Keyboard
255	088	05/08	BEGIN, BOL	Keyboard
255	096	06/00	SELECT, MARK	Keyboard
255	097	06/01	PRINT	Keyboard
255	098	06/02	EXECUTE, RUN, DO	Keyboard
255	099	06/03	INSERT, INSERT HERE	Keyboard
255	101	06/05	UNDO, OOPS	Keyboard
255	102	06/06	REDO, AGAIN	Keyboard
255	103	06/07	MENU	Keyboard
255	104	06/08	FIND, SEARCH	Keyboard
255	105	06/09	CANCEL, STOP, ABORT, EXIT	Keyboard
255	106	06/10	HELP	Keyboard
255	107	06/11	BREAK	Keyboard
255	126	07/14	MODE SWITCH, SCRIPT SWITCH, CHARACTER SET SWITCH	Keyboard
255	127	07/15	NUM LOCK	Keyboard
255	128	08/00	KEYPAD SPACE	Keyboard
255	137	08/09	KEYPAD TAB	Keyboard
255	141	08/13	KEYPAD ENTER	Keyboard
255	145	09/01	KEYPAD F1, PF1, A	Keyboard
255	146	09/02	KEYPAD F2, PF2, B	Keyboard
255	147	09/03	KEYPAD F3, PF3, C	Keyboard

Byte 3	Byte 4	Code Position	Name	Set
255	148	09/04	KEYPAD F4, PF4, D	Keyboard
255	149	09/05	KEYPAD HOME	Keyboard
255	150	09/06	KEYPAD LEFT	Keyboard
255	151	09/07	KEYPAD UP	Keyboard
255	152	09/08	KEYPAD RIGHT	Keyboard
255	153	09/09	KEYPAD DOWN	Keyboard
255	154	09/10	KEYPAD PRIOR, PAGE UP	Keyboard
255	155	09/11	KEYPAD NEXT, PAGE DOWN	Keyboard
255	156	09/12	KEYPAD END	Keyboard
255	157	09/13	KEYPAD BEGIN	Keyboard
255	158	09/14	KEYPAD INSERT	Keyboard
255	159	09/15	KEYPAD DELETE	Keyboard
255	170	10/10	KEYPAD MULTIPLICATION SIGN, ASTERISK	Keyboard
255	171	10/11	KEYPAD PLUS SIGN	Keyboard
255	172	10/12	KEYPAD SEPARATOR, COMMA	Keyboard
255	173	10/13	KEYPAD MINUS SIGN, HYPHEN	Keyboard
255	174	10/14	KEYPAD DECIMAL POINT, FULL STOP	Keyboard
255	175	10/15	KEYPAD DIVISION SIGN, SOLIDUS	Keyboard
255	176	11/00	KEYPAD DIGIT ZERO	Keyboard
255	177	11/01	KEYPAD DIGIT ONE	Keyboard
255	178	11/02	KEYPAD DIGIT TWO	Keyboard
255	179	11/03	KEYPAD DIGIT THREE	Keyboard
255	180	11/04	KEYPAD DIGIT FOUR	Keyboard
255	181	11/05	KEYPAD DIGIT FIVE	Keyboard
255	182	11/06	KEYPAD DIGIT SIX	Keyboard
255	183	11/07	KEYPAD DIGIT SEVEN	Keyboard
255	184	11/08	KEYPAD DIGIT EIGHT	Keyboard
255	185	11/09	KEYPAD DIGIT NINE	Keyboard
255	189	11/13	KEYPAD EQUALS SIGN	Keyboard
255	190	11/14	F1	Keyboard
255	191	11/15	F2	Keyboard
255	192	12/00	F3	Keyboard
255	193	12/01	F4	Keyboard
255	194	12/02	F5	Keyboard
255	195	12/03	F6	Keyboard
255	196	12/04	F7	Keyboard
255	197	12/05	F8	Keyboard
255	198	12/06	F9	Keyboard
255	199	12/07	F10	Keyboard
255	200	12/08	F11, L1	Keyboard
255	201	12/09	F12, L2	Keyboard
255	202	12/10	F13, L3	Keyboard
255	203	12/11	F14, L4	Keyboard
255	204	12/12	F15, L5	Keyboard
255	205	12/13	F16, L6	Keyboard
255	206	12/14	F17, L7	Keyboard
255	207	12/15	F18, L8	Keyboard
255	208	13/00	F19, L9	Keyboard
255	209	13/01	F20, L10	Keyboard
255	210	13/02	F21, R1	Keyboard
255	211	13/03	F22, R2	Keyboard
255	212	13/04	F23, R3	Keyboard

Byte 3	Byte 4	Code Position	Name	Set
255	213	13/05	F24, R4	Keyboard
255	214	13/06	F25, R5	Keyboard
255	215	13/07	F26, R6	Keyboard
255	216	13/08	F27, R7	Keyboard
255	217	13/09	F28, R8	Keyboard
255	218	13/10	F29, R9	Keyboard
255	219	13/11	F30, R10	Keyboard
255	220	13/12	F31, R11	Keyboard
255	221	13/13	F32, R12	Keyboard
255	222	13/14	F33, R13	Keyboard
255	223	13/15	F34, R14	Keyboard
255	224	14/00	F35, R15	Keyboard
255	225	14/01	LEFT SHIFT	Keyboard
255	226	14/02	RIGHT SHIFT	Keyboard
255	227	14/03	LEFT CONTROL	Keyboard
255	228	14/04	RIGHT CONTROL	Keyboard
255	229	14/05	CAPS LOCK	Keyboard
255	230	14/06	SHIFT LOCK	Keyboard
255	231	14/07	LEFT META	Keyboard
255	232	14/08	RIGHT META	Keyboard
255	233	14/09	LEFT ALT	Keyboard
255	234	14/10	RIGHT ALT	Keyboard
255	235	14/11	LEFT SUPER	Keyboard
255	236	14/12	RIGHT SUPER	Keyboard
255	237	14/13	LEFT HYPER	Keyboard
255	238	14/14	RIGHT HYPER	Keyboard
255	255	15/15	DELETE, RUBOUT	Keyboard

Glossary

access control list
A list, maintained by a server, of hosts from which clients are authorized to connect to the server.

active grab
A grab is active when the pointer or keyboard is actually owned by the single grabbing client.

ancestor
An ancestor of a given window is a window above it in the hierarchy.

atom
An atom is a unique ID corresponding to a string name. Atoms are used to identify properties, types, and selections.

background
A solid color or a pixmap that is used to paint portions of an *InputOutput* window that were lost or invalidated.

backing store
When a server maintains the contents of a window, the pixels saved off screen are known as a backing store.

base font name
A font name that selects a family of fonts whose members may use various character encodings. See the **XLFD** specification (included in the **X Window System (X11R6): Fonts and Text** Technical Standard).

bit gravity
The affinity of a window's contents for a specified side or corner of the window in cases where the window's size changes.

bit plane
When a pixmap or window is thought of as a stack of bitmaps, each bitmap is called a bit plane or plane.

bitmap
A pixmap of depth one.

border
An undrawable frame surrounding an *InputOutput* window. The server automatically maintains the contents of the border. Exposure events are never generated for border regions.

button grab
Buttons on the pointer may be passively grabbed by a client. When the button is pressed, the pointer is then actively grabbed by the client.

byte order
The sequence in which a stream of bytes is interpreted (such as big-endian or little-endian).

byte swapping
The process of reordering bytes in a stream of bytes when a client and server have different byte order.

character
A sequence of one or more bytes representing a single graphic symbol or control code. A character is not bound to a coded value until it is identified as part of a coded character set.

character glyph
The abstract graphical symbol for a character. Character glyphs might not map one-to-one to font glyphs, and may be context-dependent, varying with the adjacent characters. Multiple characters may map to a single character glyph.

character set
A finite set of different characters used for the representation, organization, or control of data.

charset
An encoding with a uniform, state-independent mapping from characters to codepoints (a coded character set).

There can be a direct mapping from a charset to one font, if the width of all characters in the charset is either one or two bytes. A text string encoded in an encoding such as Shift-JIS cannot be passed directly to the server, because the text imaging requests accept only single-width charsets (either 8 or 16 bits). Charsets which meet these restrictions can serve as *font charsets*. Font charsets map font indices to font glyphs, not characters to character glyphs.

A single font charset is sometimes used as the encoding of a locale; for example, ISO 8859-1.

children
A window's first-level subwindows.

client
An application program connected to a server.

clip region
The area that is painted with the window background when the window is cleared, which contains all graphics output to the window and clips any subwindows.

coded character
A character bound to a codepoint.

coded character set
A set of unambiguous rules that establishes a character set and the one-to-one relationship between each character of the set and its bit representation.

codepoint
The coded representation of a single character in a coded character set.

colormap
A set of entries that map pixel values to colors on the screen.

connection
The path between a server and a client. A client program typically (but not necessarily) has one connection to the server over which requests and events are sent.

contain, containment
A window ''contains'' the pointer if the window is viewable and the hotspot of the cursor is within a visible region of the window or a visible region of one of its inferiors. The border of the window is included as part of the window for containment. The pointer is ''in'' a window if the window contains the pointer but no inferior contains the pointer.

coordinate system
A two-dimensional space with the origin [0, 0] at the upper-left and Y increasing downward. Coordinates are integral, in terms of pixels, and coincide with pixel centers. Each window and pixmap has its own coordinate system. For a window, the origin is inside the border.

cursor
An instance of a *pointer*; in a text widget, a rendition that indicates where the next glyph will be drawn.

depth
The depth of a window or pixmap is the number of valid bit-planes used to represent a pixel.

device
Keyboards, mice, tablets, track-balls, button boxes, and so on, are all collectively known as input devices. The core X protocol only deals with two devices, the keyboard and the *pointing device*.

DirectColor
A visual class in which a pixel value is decomposed into three separate subfields for indexing. The first subfield indexes an array to produce red intensity values. The second subfield indexes a second array to produce blue intensity values. The third subfield indexes a third array to produce green intensity values. The RGB values can be changed dynamically.

display
A server, together with its screens and input devices.

drawable
Windows and pixmaps that can be used as sources and destinations in graphics operations. An *InputOnly* window cannot be used as a source or destination in a graphics operation.

encoding
A set of unambiguous rules that establishes a character set and a relationship between the characters and their representations. The character set does not have to be fixed to a finite predefined set of characters. The representations do not have to be of uniform length.

escapement
In a string, the distance in pixels in the primary draw direction from the drawing origin to the origin of the next glyph to be drawn.

event
Clients are informed of information asynchronously by means of events. These events can be generated either asynchronously from devices or as side effects of client requests.

event mask
The set of event types that a client requests relative to a window.

event propagation
The process by which device-related events propagate from the source window to ancestor windows until some client has expressed interest in handling that type of event or until the event is discarded explicitly.

event source
The window on which an event is generated.

exposure event
Events sent to clients to inform them that contents of regions of windows need to be redrawn.

extension
A feature that extends the core X protocol.

focus window
See **input focus** on page 191.

font
A set of glyphs (typically characters), with additional metric information to determine interglyph and interline spacing.

gamut
The color gamut of a device is the full range of colors that can be produced by the device.

GC, GContext
See **graphics context**.

glyph
A glyph is an image, typically of a character, in a font.

glyph image
An image of a glyph, as obtained from a glyph representation displayed on a presentation surface.

grab
Keyboard keys, the keyboard, pointer buttons, the pointer, and the server can be grabbed for exclusive use by a client. In general, these facilities are not intended to be used by normal applications but are intended for various input and window managers to implement various styles of user interface.

graphics context
Various information for graphics output such as foreground pixel, background pixel, line width, clipping region, and so on. A graphics context can only be used with drawables that have the same root and the same depth as the graphics context.

gravity
See **bit gravity** on page 187 and **window gravity** on page 196.

GrayScale
A degenerate case of *PseudoColor*, in which the red, green, and blue values in any given colormap entry are equal, thus producing shades of gray. The gray values can be changed dynamically.

Host Portable Character Encoding
The encoding of the X Portable Character Set on the host. The encoding must be the same in all locales supported by the server. If a string is said to be in the Host Portable Character Encoding, then it only contains characters from the X Portable Character Set, in the host encoding.

hotspot
The point in the cursor corresponding to the coordinates reported for the pointer.

identifier
A unique value associated with a resource that clients use to name that resource. The identifier can be used over any connection.

image
A client-side object that represents a rectangular graphic.

inferior
The inferiors of a window are all of the subwindows nested below it in the hierarchy: the children, the children's children, and so on. The term *descendant* is a synonym.

input focus
The input focus is normally a window defining the scope for processing of keyboard input. If a generated keyboard event would normally be reported to this window or one of its inferiors, the event is reported normally. Otherwise, the event is reported with respect to the focus window. The input focus also can be set such that all keyboard events are discarded and such that the focus window is dynamically taken to be the root window of whatever screen the pointer is on at each keyboard event.

input manager
Control over keyboard input is typically provided by an input manager client.

InputOnly window
A window that cannot be used for graphics requests. *InputOnly* windows are invisible and can be used to control such things as cursors, input event generation, and grabbing. *InputOnly* windows cannot have *InputOutput* windows as inferiors.

InputOutput window
An opaque window used for both input and output. *InputOutput* windows can have both *InputOutput* and *InputOnly* windows as inferiors.

internationalization
The process of making software adaptable to the requirements of different native languages, local customs, and character string encodings. Making a computer program adaptable to different locales without program source modifications or recompilation.

ISO 2022
ISO standard for code extension techniques for 7-bit and 8-bit coded character sets.

key grab
Keys on the keyboard can be passively grabbed by a client. When the key is pressed, the keyboard is then actively grabbed by the client.

keyboard grab
A client can actively grab control of the keyboard, and key events are sent to that client rather than to the client the events would normally have been sent to.

keysym
An encoding of a symbol on a keycap on a keyboard.

Latin-1
The coded character set defined by the ISO 8859-1 standard.

Latin Portable Character Encoding
The encoding of the X Portable Character Set using the Latin-1 codepoints plus ASCII control characters. If a string is said to be in the Latin Portable Character Encoding, then it only contains characters from the X Portable Character Set, not all of Latin-1.

locale
The definition of the subset of a user's environment that depends on language and cultural convention. In this Technical Standard, the current locale is the current setting of the LC_CTYPE *setlocale* category. The current locale affects:

- Encoding and processing of input method text
- Encoding of resource files and values
- Encoding and imaging of text strings
- Encoding and decoding for inter-client text communication

localization
The process of establishing information within a computer system specific to the operation of particular native languages, local customs, and coded character sets.

mapped
A window is said to be mapped if a map call has been performed on it. Unmapped windows and their inferiors are never viewable or visible.

modifier keys
Shift, Control, Meta, Super, Hyper, Alt, Compose, Apple, CapsLock, ShiftLock, and similar keys are called modifier keys.

monochrome
A special case of *StaticGray* in which there are only two colormap entries.

multibyte character
A sequence of one or more bytes representing a single graphic symbol or control code. This term corresponds to the same term in the ISO C standard, where a single-byte character is a special case of a multi-byte character. Unlike the usage in the ISO C standard, *character* here has no necessary relationship with storage space, and *byte* is used when storage space is discussed.

obscure
Window A obscures window B if both are viewable *InputOutput* windows, A is higher in the global stacking order, and the rectangle defined by the outside edges of A intersects the rectangle defined by the outside edges of B. Window borders are included in the calculation. A window can be obscured and yet still have visible regions. (See **occlude**.)

occlude
Window A occludes window B if both are mapped, A is higher in the global stacking order, and the rectangle defined by the outside edges of A intersects the rectangle defined by the outside edges of B. Window borders are included in the calculation. (See **obscure**.)

padding
Bytes inserted in the data stream to maintain alignment of the protocol requests on natural boundaries. This increases ease of portability to some machine architectures.

parent window
If C is a child of P, then P is the parent of C.

passive grab
Grabbing a key or button is a passive grab. The grab activates when the key or button is actually pressed.

pixel value
An N-bit value, where N is the number of bit planes used in a particular window or pixmap (that is, N is the depth of the window or pixmap). For a window, a pixel value indexes a colormap to derive an actual color to be displayed.

pixmap
Off-screen resources that are used for various operations, such as defining cursors as tiling patterns or as the source for certain raster operations. A pixmap is a three-dimensional array

of bits, normally thought of as a two-dimensional array of pixels, where each pixel can be a value from 0 to (2^N)–1 and where N is the depth (Z axis) of the pixmap. A pixmap can also be thought of as a stack of N bitmaps.

plane
When a pixmap or window is thought of as a stack of bitmaps, each bitmap is called a plane or bit plane.

plane mask
A bit mask describing which planes are to be modified by graphics operations.

pointer
The graphical representation indicating the location of the pointing device.

pointer grab
A client can actively grab control of the pointer. Button and motion events are then sent to that client rather than to the client to which the events would normally have been sent.

pointing device
A pointing device is typically a mouse, tablet, or some other device with effective dimensional motion. There is only one visible cursor defined by the core protocol, and it tracks whatever pointing device is attached as the pointer.

POSIX
Portable Operating System Interface, ISO / IEC 9945-1 (IEEE Std. 1003.1).

POSIX Portable Filename Character Set
The set of 65 characters which can be used in naming files on a POSIX-compliant host that are correctly processed in all locales. The set is:

a..z A..Z 0..9 ._-

property
Windows may have associated properties, which consist of a name, a type, a data format, and some data. The protocol places no interpretation on properties. They are intended as a general-purpose naming mechanism for clients. For example, clients might use properties to share information such as resize hints, program names, and icon formats with a window manager.

property list
The list of properties that have been defined for a window.

PseudoColor
A class of colormap in which a pixel value indexes the colormap to produce independent red, green, and blue values. That is, the colormap is viewed as an array of triples (RGB values). The RGB values can be changed dynamically.

redirecting control
Window managers (or client programs) may want to enforce window layout policy in various ways. When a client attempts to change the size or position of a window, the operation may be redirected to a specified client rather than the operation actually being performed.

renderer
Code that takes font data in its raw format and converts it to the font server's format.

reply
Information requested by a client is sent back to the client with a reply. Both events and replies are multiplexed on the same connection. Most requests do not generate replies,

although some requests generate multiple replies.

request
A command to the server; a single block of data sent over a connection.

resource
Windows, pixmaps, cursors, fonts, graphics contexts, and colormaps are known as resources. All have unique identifiers associated with them for naming purposes. The lifetime of a resource usually is bounded by the lifetime of the connection over which the resource was created.

RGB values
Red, green, and blue (RGB) intensity values are used to define color. These values are always represented as 16-bit unsigned numbers, with 0 being the minimum intensity and 65535 being the maximum intensity. The server scales the values to match the display hardware.

root window
The common ancestor of all windows for a screen. A root window has no parent.

save-set
A client's list of other clients' windows that, if they are inferiors of one of the client's windows at connection close, should not be destroyed and should be remapped if currently unmapped. Save sets are typically used by window managers to avoid lost windows if the manager terminates abnormally.

scanline
A list of pixel or bit values viewed as a horizontal row (all values having the same y coordinate) of an image, with the values ordered by increasing X coordinate.

scanline order
Scanlines ordered by increasing Y coordinate.

screen
A bounded displayable area that houses the root window. A server can provide several independent screens, which typically have physically independent monitors.

selection
A selection can be thought of as an indirect property with dynamic type; that is, rather than having the property stored in the server, it is maintained by some client (the ''owner''). A selection is global in nature and is thought of as belonging to the user (although maintained by clients), rather than as being private to a particular window subhierarchy or a particular set of clients. When a client asks for the contents of a selection, it specifies a selection ''target type''. This target type can be used to control the transmitted representation of the contents. For example, if the selection is ''the last thing the user clicked on'' and that is currently an image, then the target type might specify whether the contents of the image should be sent in XY format or Z format. The target type can also be used to control the class of contents transmitted; for example, asking for the ''looks'' (fonts, line spacing, indentation, and so on) of a paragraph selection rather than the text of the paragraph. The target type can also be used for other purposes, but such semantics are unspecified.

server
The server provides the basic windowing mechanism. It handles connections from clients, multiplexes graphics requests onto the screens, and demultiplexes input back to the appropriate clients.

server grab
The server can be grabbed by a single client for exclusive use. This prevents processing of any requests from other client connections until the grab is completed. This is typically only a transient state for such things as rubber-banding, pop-up menus, or to execute requests indivisibly.

shift sequence
Control characters and escape sequences which temporarily (single shift) or permanently (locking shift) cause a different character set to be in effect (''invoking'' a character set).

sibling
A child of the same parent window.

stacking order
The relationship between sibling windows that determines whether some windows obscure or occlude others.

state-dependent encoding
An encoding in which an invocation of a charset can apply to multiple characters in sequence. A state-dependent encoding begins in an ''initial state'' and enters other ''shift states'' when specific ''shift sequences'' are encountered in the byte sequence.

state-independent encoding
Any encoding in which the invocations of the charsets are fixed, or span only a single character.

StaticColor
A degenerate case of *PseudoColor* in which the RGB values are predefined and read-only.

StaticGray
A degenerate case of *GrayScale* in which the gray values are predefined and read-only. The values are typically linear or near-linear increasing ramps.

status
The return value of many functions.

stipple
A bitmap used to tile a region that may also serve as an additional clip mask for a fill operation with the foreground color.

tile
A pixmap can be replicated in two dimensions to tile a region. The pixmap itself is also known as a tile.

timestamp
A time value expressed in milliseconds. It typically is the time since the last server reset. Timestamp values wrap around (after about 49.7 days).

top-level
A child of the root window.

TrueColor
A degenerate case of *DirectColor* in which the subfields in the pixel value directly encode the corresponding RGB values; that is, the colormap has predefined read-only RGB values. The values are typically linear or near-linear increasing ramps.

type
An arbitrary atom used to identify the interpretation of property data. Types are completely uninterpreted by the server and are solely for the benefit of clients.

unviewable
A window is unviewable if it is mapped but some ancestor is unmapped. See also **viewable**.

viewable
A window is viewable if it and all of its ancestors are mapped. This does not imply that any portion of the window is actually visible. Graphics requests can be performed on a window when it is not viewable, but output is not retained unless the server is maintaining backing store.

visible
A region of a window is visible if someone looking at the screen can actually see it; that is, the window is viewable and the region is not occluded by any other window.

whitespace
Any spacing character. On implementations that conform to the ISO C library, whitespace is any character for which the *isspace*( ) function returns true.

window
An area (usually rectangular) on the screen that holds graphical output.

window gravity
When windows are resized, subwindows may be repositioned automatically relative to some position in the window. This attraction of a subwindow to some part of its parent is known as window gravity.

window manager
A client that manipulates windows on the screen and typically provides the user interface policy.

X Portable Character Set
A basic set of 97 characters which are assumed to exist in all locales supported by this Technical Standard. This set contains the following characters:

a..z A..Z 0..9 !"#$%&’()*+,-./:;<=>?@[\]^_‘{|}~ <space>, <tab>, and <newline>

This is the left/lower half (also called the G0 set) of the graphic character set of ISO 8859-1 plus <space>, <tab>, and <newline>. It is also the set of graphic characters in 7-bit ASCII plus the same three control characters. The actual encoding of these characters on the host is implementation-dependent; see **Host Portable Character Encoding** on page 190.

XDCCC
The X Device Color Characterization Convention, a method of specifying device-independent color spaces.

XLFD
The X Logical Font Description Conventions that define a standard syntax for structured font names.

XY Format
The data for a pixmap is said to be in XY format if it is organized as a set of bitmaps representing individual bit planes, with the planes appearing from most-significant to least-significant in bit order.

Z Format

The data for a pixmap is said to be in Z format if it is organized as a set of pixel values in scanline order.

Index